A Software Architecture for Seamless Simulated to Real Testbed Transition for O-RAN AI

A Software Architecture for Seamless Simulated to Real Testbed Transition for O-RAN AI

J. L. Herrera[§], N. Villegas[*], L. Diez[*], D. Scotece[‡], L. Foschini[‡] and R. Agüero[*]
[§] Distributed Systems Group, TU Wien, Austria. e-mail: j.gonzalez@dsg.tuwien.ac.at
[*] Universidad de Cantabria, Spain. e-mail: villegasn@unican.es, {ldiez, ramon}@tlmat.unican.es
[‡] Università di Bologna, Italy. e-mail: {domenico.scotece, luca.foschini}@unibo.it

***Abstract*—The growing interest in leveraging the potential of open interfaces and standards brought forth the Open Radio Access Network (O-RAN) specifications, which are becoming increasingly relevant in 5G and 6G Radio Access Networks (RANs). Amongst other features, O-RAN is expected to bring intelligence to 5G and 6G networks by enabling native support for Artificial Intelligence (AI) in the management of the RAN and its control plane. However, AI modules need to be previously trained and evaluated in simulated or emulated network testbeds as not to negatively affect users in real RANs. This separation between initial training and evaluation and the final usage of AI in real network deployments brings many challenges: while the O-RAN specification expects these AI modules to be reusable across RANs, their interfaces, timing, and data models are coupled with those of the RAN, requiring significant re-development effort to make the transition from simulation to real usage. To address this challenge, this work proposes the Open Intelligent Interfaces and Infrastructure Architecture (OI3A), a software architecture to decouple the specifics of RANs and simulators from the AI modules that are trained, evaluated, and used in them. OI3A is evaluated in a realistic proof-of-concept, which considers a use case of Deep Reinforcement Learning-driven MAC scheduling, allowing online training with minimal overhead.**

***Index Terms*—O-RAN, AI, Software Architecture, Simulation**

I. Introduction

Traditionally, Radio Access Networks (RANs) have been deployed using a closed, black-box model that relies on proprietary hardware, software, and interfaces [1]. This design limited flexibility, innovation, and vendor diversity. In response, Open Radio Access Network (O-RAN) represents a fundamental change in the development and operation of mobile networks by embracing open and modular architectures. Thus, this revolutionary paradigm promotes seamless interoperability, while increasing control and openness in the development and deployment of RAN components.

The O-RAN architecture defines two different types of controller: i) Near-RT RAN Intelligent Controller (RIC), responsible for tasks with tight latency constraints; and ii) Non-RT RIC, which manages functions with more relaxed timing requirements [2]. The RIC is a software-defined component that interacts with the RAN to collect data, automate and optimize RAN operations. A key feature of the RIC architecture is its support for third-party applications: near-real-time functions, known as *xApps*, and non-real-time functions, *rApps*.

Deploying AI-based solutions in real network environments presents significant challenges, particularly in the context of data collection and model training. Unlike simulated setups, real networks offer limited flexibility and observability. This makes it difficult to gather high-quality, comprehensive datasets without disrupting ongoing services. These challenges are amplified when using feedback-based learning methods such as DRL, which requires continuous interaction with the environment to explore, learn, and adapt. During the early stages of training, these agents often operate suboptimally, potentially degrading network performance or failing to meet QoS constraints. Therefore, safe and efficient exploration is critical, especially in production environments where service reliability is paramount. As a result, extensive offline pre-training, realistic emulation, or sandboxed deployments often become essential before safely integrating these AI agents into real-world systems.

However, even realistic simulation and emulation present a challenge when AI agents are finally leveraged in real testbeds, especially when they are deployed as xApps. As xApps need to communicate with their corresponding near-RT RIC, they must adhere to the RIC-specific protocols and data formats it specifies. This phenomenon forces xApps to be coupled with their corresponding near-RT RIC [3], which hinders the compatibility of an xApp to be moved across near-RT RICs. This coupling is especially problematic when tackling the transition from simulated to real systems, even considering emulated testbeds, as they usually deploy different near-RT RICs [3]. Simulated testbeds, which bring potential benefits due to their ability to reproduce a wide variety of scenarios at minimal cost [4], [5], can be especially problematic, as they often have their own simulated near-RT RIC, requiring, in turn, simulated xApps, implemented as part of the simulator. Simulated xApps are also often coupled to simulation timing, which is not necessarily real-time, and can thus cause further interoperability issues when deployed in real testbeds. Similar issues arise for service models: if the near-RT RIC has xApps directly leveraging the same E2 Service Model used at the RAN level, they may need to be adapted to new E2 Service Models, or new versions of them, as supported by the RAN, even if the RIC and xApp do not change. Hence, applying AI at the xApp level often requires porting, through significant code and interface adaption, as well as re-development, when addressing the transition between simulated and real testbed. This is also true when AI xApps, that are already in production in real testbeds, need to be compared in simulations against

new proposals, as the coupling remains.

To address this need in the design, deployment and evaluation of O-RAN networks, this work proposes the Open Intelligent Interfaces Infrastructure Architecture (OI3A), an architecture to decouple AI models from near-RT RICs and the underlying RANs. OI3A eases the transition from simulated to real testbeds by separating the concerns of RIC-specific protocols, data formats, and E2 Service Models behind a single *connector* xApp, which AI models can consume directly, minimizing the need for re-development and enhancing reusability. Specifically, the contributions of this paper are:

- The proposal of decoupling the protocols between the near-RT RIC, the xApps, and the AI modules of O-RAN networks.
- The proposal of OI3A as an architecture to achieve said decoupling. OI3A has different design variants to ensure it can adapt to different AI use cases.
- The evaluation of OI3A using a DRL-based agent in the 5G-LENA simulator. OI3A proves to have minimal overhead while enabling the DRL agent to be trained online during a simulation.

The rest of the paper is structured as follows. Section II analyzes related works in AI for O-RAN and software architectures to support it. Section III presents the proposed OI3A architecture and describes the modules and interactions between them. Section IV evaluates an OI3A implementation using a DRL agent and the 5G-LENA simulator. Finally, Section VI concludes the paper and presents future research directions.

II. Related works

Recently, there has been growing interest in leveraging Artificial Intelligence (AI) for 5G New Radio (NR). In [6], the authors propose a framework that integrates Lyapunov optimization with Deep Reinforcement Learning (DRL) to enable online computation offloading in mobile-edge computing networks. Similarly, [7] introduces a Reinforcement Learning (RL)-based scheduler capable of selecting among different resource allocation strategies to minimize latency and packet loss, thereby meeting Quality of Service (QoS) requirements. The study concludes that relying on a single scheduling policy is insufficient for achieving optimal performance under dynamic network conditions and diverse traffic demands.

Recent developments in AI research have mainly focused on resource management strategies at higher network layers, where latency in decision-making is more tolerable [8]. Applying such algorithms to lower-layer resource control and slot-level scheduling remains challenging due to stringent timing requirements and limited or outdated feedback availability. For example, [9] presents a random forest-based model that predicts user modulation schemes and MCS, thereby eliminating the dependency on traditional Channel State Indicator (CSI) feedback mechanisms. Another example is [10], which employs a Machine Learning (ML) model to forecast buffer occupancy and Channel Quality Indicator (CQI) values in the MAC scheduler using historical data. This work advocates for a one-time resource allocation approach over specific time intervals through Prediction Based Scheduling (PBS), aiming to conserve Physical Downlink Control Channel (PDCCH) resources and minimize allocation delays. However, this method assumes channel stability throughout the scheduled duration, which may not always hold true. Moreover, traditional RL techniques such as Q-Learning and Policy Gradient often struggle with scalability, particularly as the number of active users grows, and the discrete action space becomes increasingly large and complex [11].

The interest in leveraging simulators for training and evaluating AI models before integrating them into real networks has also led to proposals in this aspect. Most interestingly, Gawłowicz and Zubow proposed ns3-gym [5], a complete framework to integrate DRL model training and evaluation into ns-3 network simulator. Specifically, ns3-gym allows developers to connect OpenAI Gymnasium, the *de facto* standard tool for building DRL environments, with ns-3. This connection is done by implementing a specific interface on two ends: ns-3, written in C++, and the DRL agent using Gymnasium, written in Python. This is achieved by the ns3-gym module, which integrates with the ns-3 core, and serves as a C++ interface to Gymnasium. Internally, ns3-gym deploys both the ns-3 simulation and a separate Gymnasium agent, which are communicated using ZeroMQ and share data using Protocol Buffers. Another interesting proposal is ns3-ai [4] by Yin *et al.*. Conceptually, ns3-ai is similar to ns3-gym, as it enables the integration of DRL, as well as other AI systems, with the ns-3 simulator for online training and evaluation. The main differences are that ns3-ai supports AI frameworks outside Gymnasium, such as TensorFlow or PyTorch, and that the communication is not made through network protocols, but directly through a shared memory pool. Both ns3-gym and ns3-ai are interesting approaches, but they all are inherently coupled with the ns-3 simulator. This coupling can be limiting for the transition of these AI models from simulated network testbeds for training and initial evaluation into real testbeds and networks.

An important benefit of the O-RAN ecosystem is its reusability and portability, as open interfaces enable the software developed for one RAN to be applicable in other RANs. This concept of reusability has also motivated its application as a design guideline not only to individual modules in the O-RAN architecture (e.g., xApps), but also to the internal architecture within such modules. One interesting example is ColO-RAN [12], proposed by Polese *et al.* as an architecture to develop DRL modules for O-RAN. ColO-RAN introduces, on the one hand, a complete reusable architecture for DRL in real testbeds and, on the other hand, an architecture to integrate DRL agents into xApps. In a similar line, the Intelligent xApp Architecture (IxAA) [3] is a general architecture for xApps that divides them into multiple, reusable and pluggable modules, aimed at developing a single xApp that can then be reused by a variety of AI modules with different objectives, with minimal to no changes to the code. These architectures were inspirational for OI3A, but have a different focus. ColO-

RAN's xApp architecture, as well as IxAA, focus on the xApp architecture, i.e., they aid xApp developers to integrate AI models into them. OI3A instead makes reusable components to aid AI model developers to easily integrate them in O-RAN testbeds. Moreover, neither IxAA nor ColO-RAN address the transition from simulated to real testbeds, a key tenet of OI3A.

Overall, OI3A is in line with the existing works in bringing AI to 5th Generation (5G) networks, and O-RAN networks specifically, as well as with the integration of AI with simulators, and the corresponding reusability aspects. Nonetheless, to the best of our knowledge, OI3A is the first solution to address the gap of seamless transition from simulated to real testbeds by leveraging reusable modules that provide the same interface to AI models, contributing to the provision of intelligence into O-RAN deployments.

III. Proposed architecture: OI3A

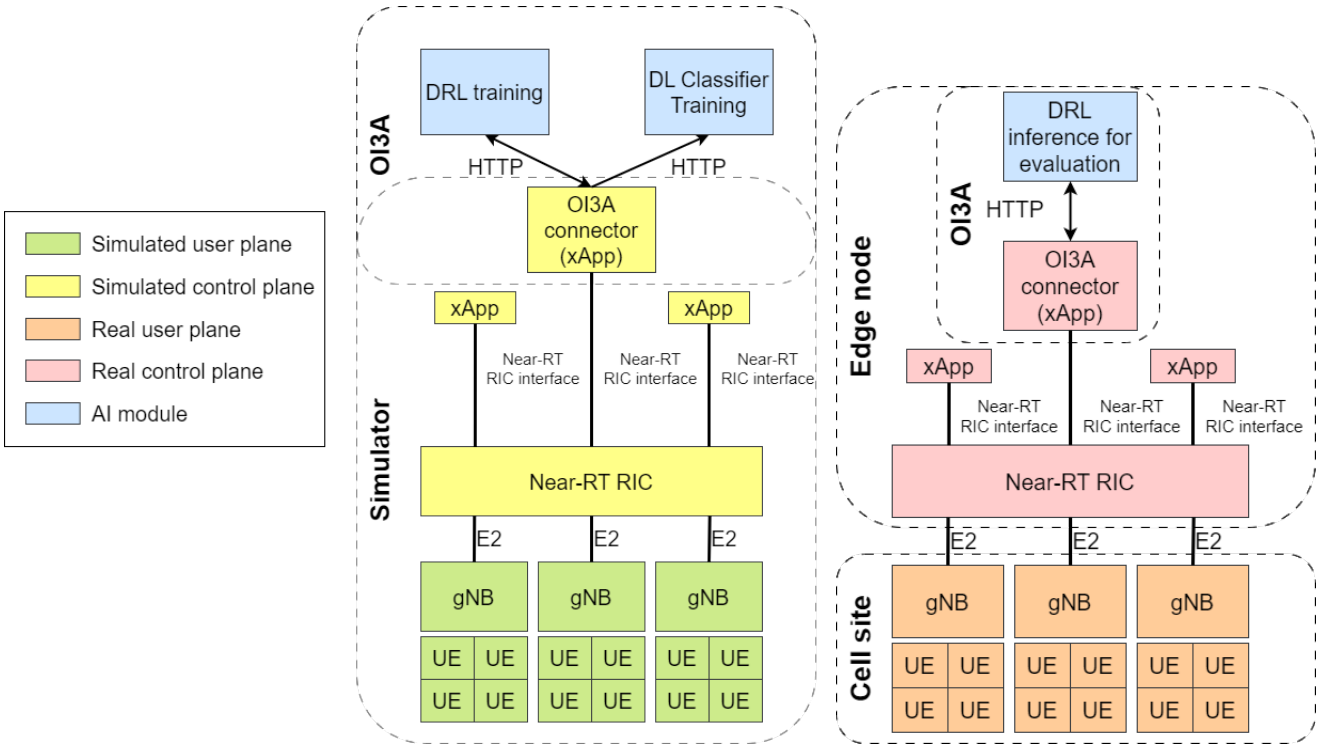
To address these needs in AI for O-RAN, it is necessary to design a software architecture that eases the task for developers. This architecture must, on the one hand, allow for AI models to be used in both simulated and real testbeds as seamlessly as possible, and on the other hand, support different AI proposals that interact differently with the RAN, including synchronous and asynchronous communications, as well as subscriptions, data collection, and control actions. This section introduces OI3A, our proposed solution to address such requirements. We depict the architecture itself and provide details on its support for different kinds of interactions.

The first feature of OI3A is that we consider two different design variants of its architecture, depending on the interaction style that developers wish to use. We consider a *synchronous* design variant, meant for interactions in which the AI must await data from the RAN after performing a request. This variant is especially devoted to Reinforcement Learning-based systems, in which the agent must wait for an update in its observation to take an action, and must then await the reward provided as feedback. In contrast, OI3A also includes an *asynchronous* design variant, in which interactions are governed by a message-oriented middleware. Using the asynchronous variant, AI systems interact with the RAN by subscribing to topics on the metrics they wish to receive updates on, and can perform actions on the RAN by sending their own messages to the appropriate topics. This design is more appropriate for traditional AI models, whose predictions can adjust configuration parameters of the RAN without requiring synchronization or explicit feedback from the RAN itself. Both variants are considered in OI3A, and they can be implemented using different techniques: protocols, remote procedure call middlewares, or message brokers. Nonetheless, due to their widespread use among developers and the vast amount of tools to support their use, we recommend implementing the synchronous design variant leveraging REST (i.e., stateless HTTP and JSON), and MQTT with JSON for the asynchronous variant. From now on, we will assume that the discussed implementations are based on these technologies.

The overall architecture is depicted in Fig. 1, in an example scenario with 3 gNBs and 2 external xApps, both in a simulated and in a real testbed. From an architectural perspective, the differences between the synchronous REST and asynchronous MQTT variants are minimal, only affecting the protocol used by the AI modules to enable the required interactions, and only the synchronous variant is represented for the sake of clarity. The overview of the architecture begins, using Fig. 1a as a point of reference, in bottom-up order. At the bottom, the UE and gNB modules represent the O-RAN user plane, which, in this particular case, is fully simulated, serving 6 simulated UEs through 3 simulated gNBs. The aggregation or disaggregation of the gNBs, as well as the functional split, depends on the specific scenario and support from the simulation framework, but they do not impact whatsoever OI3A. The gNBs connect to the near-RT RIC using the E2 interface, as specified in O-RAN. Most simulators do not support connections to external near-RT RICs, and they directly include them [13], [14] within their implementations. While OI3A can also support external near-RT RICs (as shown in Fig. 1b), this specific scenario assumes the near-RT RIC is simulated. Moreover, two xApps, unrelated to OI3A, are under execution. These xApps show that OI3A can be used regardless of the execution of other xApps in the scenario.

The third xApp is the first OI3A module, the *Connector*. The OI3A connector is an xApp which communicates with the near-RT RIC by means of its specific interface (e.g., Redis, HTTP), like other xApps. Moreover, as the xApps are simulated in this scenario, the OI3A connector is also implemented as a simulated xApp (e.g., within ns-3). The crucial role of the connector is to implement an API that allows the communication between AI modules and the RAN. In Fig. 1a, the connector implements a REST API. The API must include procedures for commonplace O-RAN processes, such as subscribing to metrics, pushing updates of those metrics to the subscribed modules, and executing actions on the RAN. Although the connector is simulated, it exposes a real REST API to the outside that can be thus accessed by other software. Ideally, a separate set of threads or a different process should be used for the connector, ensuring that, while still part of the simulation, it can respond as an API.

On top of the connector there are a number of AI modules, which are entirely real. The example that is illustrated in Figure 1a has two AI modules, one for training DRL-based models, and one for training Deep Learning-based classifiers. In both cases, these AI modules should include a callback mechanism for receiving updates on specific metrics. In this case, each module implements its own REST API, with a single endpoint for receiving updates. Hence, each AI module can subscribe, at the connector, to certain metrics, sending their own callback URL as part of their subscription message, allowing the connector to push messages to them. Moreover, AI modules can send messages to the connector through its REST API to execute necessary actions in the RAN. Communications between the connector and AI modules are based on HTTP and JSON, using the object definitions specified in the



(a) Training a DRL and a classifier model in a simulator.

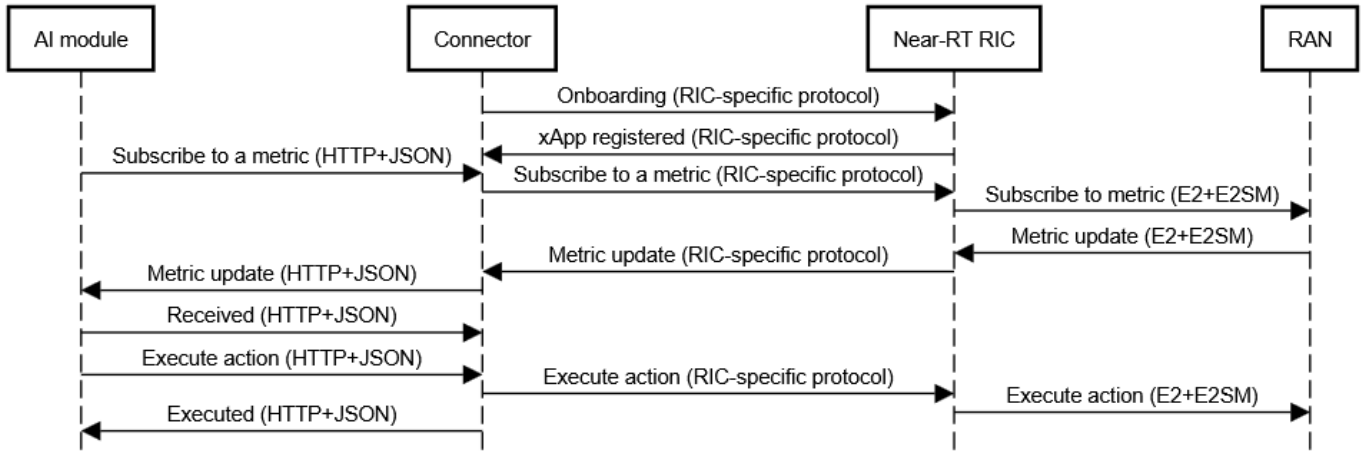
(b) Evaluating a DRL model in a real testbed.

Fig. 1: Software architecture of an OI3A example scenario with 3 gNBs and 2 external xApps using a synchronous REST variant.

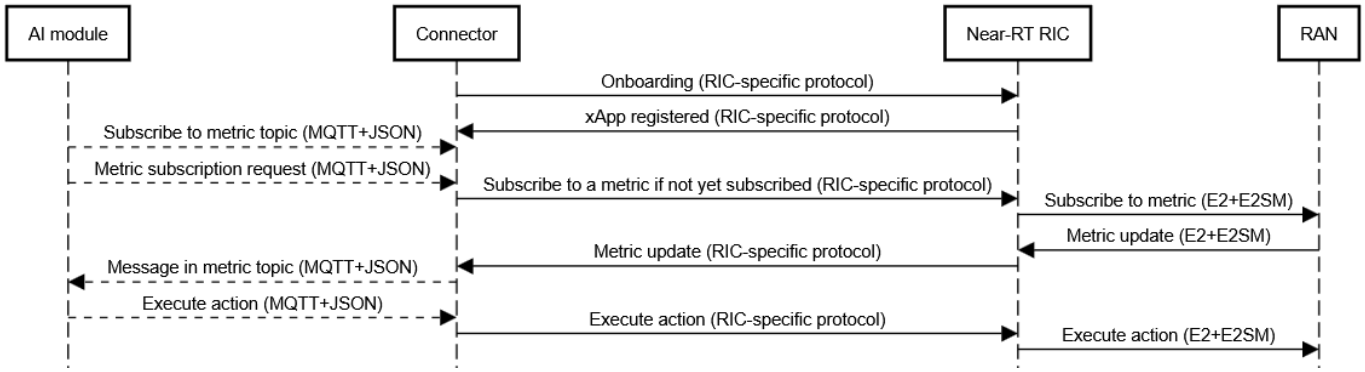
APIs. The AI modules, hence, do not need to be concerned with the specific service model, the encoding used by the near-RT RIC, nor particularities about the near-RT RIC interface, as the connector handles it for them. Moreover, AI modules should be implemented to be as reusable as possible, for instance allowing the training of multiple models that use different metrics.

To better understand the communications in OI3A, sequence diagrams are provided in Fig. 2, highlighting the protocols and data formats that are used. Starting with the synchronous variant implemented with REST APIs (Fig. 2a), the first interaction is performed by the connector upon instantiation. The connector sends a message to the near-RT RIC requesting onboarding as an xApp, which the near-RT RIC allows. This interaction, as all interactions between the connector and the near-RT RIC, are governed by the protocols and data formats specified by the near-RT RIC. Once the connector is running, AI modules can be instantiated. In this case, the AI module first subscribes to a metric. This subscription is expressed by a simple HTTP POST message, which carries the relevant information in its body, using JSON format. The connector then parses this subscription request into RIC-specific semantics (i.e., protocols and data formats), sending it to the the near-RT RIC, which will then perform the subscription in the RIC, leveraging the E2 interface and the E2 service model the RAN supports for the subscription (e.g., E2SM-KPM v1.2 in ASN.1 format). In parallel, the connector notifies the AI module of the successful subscription with an HTTP 200 message, with the necessary feedback information also in JSON format. Besides, HTTP status codes can also be used to communicate different errors (e.g., 404 for non-existent metrics, 501 for unimplemented metrics, 403 if the connector is not authorized to subscribe to the metric). Once the RAN has an update on the metric, it will relay the relevant KPM information to the near-RT RIC using the same E2 interface and service model as before. This update will be pushed, via RIC-specific protocols, to the connector, as, to the eyes of the near-RT RIC, it is the xApp that subscribed to the metric. The connector will then notify the module of an update by performing a callback to its own REST API, with another HTTP POST message that forwards the new metric data in the appropriate JSON schema. The AI module will respond with an HTTP 200 message to acknowledge it has correctly received the update. The AI module then uses this metric update to perform an action, sending an execute action (HTTP PUT) message to the connector, with the relevant changes and actions codified with JSON. This is relayed to the near-RT RIC, and then to the RAN. The connector will finally respond the agent with the appropriate HTTP status code.

The asynchronous variant has a slightly different sequence diagram, illustrated in Fig. 2b, which uses MQTT and JSON formats. Interestingly, the communications between the connector and the near-RT RIC, as well as between the near-RT RIC and the RAN, are identical in both synchronous



(a) Synchronous REST variant.



(b) Asynchronous MQTT variant.

Fig. 2: Sequence diagram of OI3A interactions.

and asynchronous variants. As changes are only introduced in the OI3A components, namely the connector and the AI module, the interface across the rest of the elements are left unchanged. Following the same process that was described for the synchronous variant, the connector first completes the onboarding process. Then, the AI module, via MQTT, subscribes to the topic of the metric it wants to consume. This subscription is performed at the MQTT broker, which is embedded as an entity of the connector in OI3A. If the module is interested in multiple metrics, it can subscribe to as many topics as necessary. Then, it sends a message to a reserved subscription request MQTT topic, providing a JSON list of the metrics the AI module has subscribed to. The connector, one by one, sends the appropriate subscription requests to the near-RT RIC, which are then forwarded to the RAN. As multiple AI modules can be interested in the same metric, these requests are only sent if the connector is not yet subscribed to the metric. Then, as metric updates are received in the connector from the RAN, passing through the near-RT RIC, the connector simply relays them to the appropriate topics in JSON format. These messages are received by all AI modules subscribed to those topics via MQTT. For action execution, the structure is similar: a reserved topic exists for AI modules to request actions to be executed on the RAN. The connector is subscribed to this topic, so actions sent by AI modules are parsed, and the relevant actions are executed by messaging the near-RT RIC and, in turn, the RAN. One distinct feature of asynchronous messaging is that it is non-blocking, and so AI modules are assumed to not await feedback, which is thus not provided, unlike in the synchronous variant.

OI3A provides various benefits to AI in O-RAN. Most interestingly, the architecture decouples AI models from the RAN in multiple aspects, enabling AI models to be moved not only from a simulated testbed to a real one, but also across simulated and real testbeds. First, OI3A decouples the data model offered to AI modules, which is the JSON schema defined at the connector, from the service model at the RAN level, which is tied to an E2 Service Model. This decoupling brings relevant advantages, as it allows for compatibility with multiple service models through various topics (e.g., messages to be sent using E2SM-KPM v1.0 can be sent to the topic OI3A/INFO/V1, while those that leverage E2SM-KPM v1.2 can be sent to OI3A/INFO/V1-2) or API versioning (e.g., E2SM-KPM v1.0 and v1.2 APIs can be made accessible at HOST/API/V1 and HOST/API/V1-2, respectively). Backwards compatibility can also be managed by OI3A by making the

connector automatically convert across versions (e.g., if the RAN only supports E2SM-KPM v1.2, the connector can transform data sent to the API for E2SM-KPM v1.0 to the v1.2 format). OI3A can also leverage a common data representation for multiple service models, enabling the same JSON schema to be used in the AI modules, regardless of the underlying service model. This decoupling is useful when the simulated and real near-RT RICs have different service models (e.g., the simulated one uses a custom, non-standard service model while the real one uses a standard one) or differently encoded service models (e.g., Protocol Buffers and ASN.1).

OI3A also decouples timings between AI modules and the RAN, which is especially important for the simulated to real testbed transition. Simulators, by their own nature, do not run in real time, and they can be faster or slower, depending on their computational load, the machine they run on, and the particular scenario they implement. Leveraging OI3A, AI modules only have to be concerned about when new data is available, either when a new message is sent to their corresponding topic if the asynchronous variant is employed, or when they receive a callback in the synchronous case. Reusability is also interesting: only the OI3A connector needs to be developed for each simulated or real testbed in which the AI-based system is to be integrated. Connectors do not have to be built from scratch, they just need to be developed for the particular characterstics of a testbed, and can be reused for testbeds with the same characteristics (e.g., a single connector for 5G-LENA can be reused for different AI modules to be used with 5G-LENA). There are also benefits to its implementation using widespread technologies such as HTTP, JSON, and MQTT. For example, in the synchronous variant, the APIs can be defined using the OpenAPI specification [15] to support automated generation and documentation of server and client code. Most interestingly, this allows for the automated generation of the necessary code for implementing AI modules, including key components of architectures such as IxAA [16]. Last, security and authorization can be handled using the same mechanisms leveraged by other HTTP or MQTT solutions, such as HTTPS or JSON Web Tokens [17].

IV. Evaluation

This section presents some results, based on simulation, which validate the proposed architecture, considering a DRL use case. Specifically, Section IV-A depicts the scenario in which OI3A was evaluated, while Section IV-B discusses the obtained results.

A. Evaluation setup

The simulations have been conducted on a server with an Intel® Xeon® E3-1241 v3 processor (4 cores, 8 threads) running at 3.50 GHz, with 32 GB of DDR3 RAM (4 × 8 GB, 1600 MHz), under Ubuntu 20.04.6 LTS.

The evaluation scenario comprises a single gNB and four UEs, all located within the same beam. Simulations of the RAN are performed using ns-3 5G-LENA, a comprehensive framework that offers accurate and detailed modeling of the 5G NR PHY and MAC layers. All UEs generate mixed 3GPP eXtended Reality (XR) traffic: two UEs are configured with Virtual Reality (VR) services, while the remaining two generate Cloud Gaming (CG) traffic. The VR traffic primarly comprises high-bandwidth and low-latency video streams for inmersive virtual environments, characterized by high resolution and stringent delay requirements. Meanwhile, the CG traffic models workloads involving computation offloading to a remote server, where low-latency and reliable connectivity between the UE and cloud are critical. The VR and CG traffic models differ in their statistical characteristics, reflecting distinct patterns in packet sizes, inter-arrival times, and traffic burstiness, leading to challenging demands on the RAN.

The objetive is to satisfy the specific QoS requirements of each XR UE while minimizing the consumption of radio resources to promote an efficient use of the available spectrum. To achieve this, a QoS-aware MAC scheduler [18] handles the Resource Block (RB) allocation. In addition, an AI module is deployed to enhance the scheduling process. Specifically, a Proximal Policy Optimization (PPO) agent is trained online at the AI module to optimize the MAC scheduler performance by tuning its configurable parameters. These parameters control the relative importance (weight) assigned by the scheduler to three key objectives: stabilizing traffic queues, meeting QoS requirements and minimizing RB usage. The agent periodically receives configuration requests for the scheduler from the RAN via the OI3A connector, while continuously learning and refining its policy to meet the stringent QoS demands of XR services and maximize spectral efficiency. In particular, a constant period of 2 seconds of simulated time is consdered for the weights updates. Communication between the emulated RAN (5G-LENA) and the AI module is established via a REST API, which implements the synchronous variant of the OI3A connector.

B. Evaluation results

Figure 3 illustrates the reward evolution over time, comparing the DRL-based approach, both during training and after being trained, with a random strategy, in which the weights of the scheduler are randomly selected. As can be seen, the untrained agent shows a clear learning trend, outperforming the random agent after around 1500 steps, and gradually converging to a reward close to 0.6 by step 6000. In contrast, the trained agent achieves a stable reward of 0.6 already at the beginninng, with much lower variation. These results evince that the use of the AI module yields a clear peformance improvement, since the reward consistently stays above the one that was obtained with the random policy. Moreover, the results validate the approach of using simulation-based training as a reliable method for building models ready for deployment, and successfully assess the feasilibility of the proposed system design, which allows the models to be directly applied to real systems.

In order to assess the overhead induced by the proposed solution, Figure 4 illustrates the response times that were observed across 1000 steps. It is worth noting that the analysis

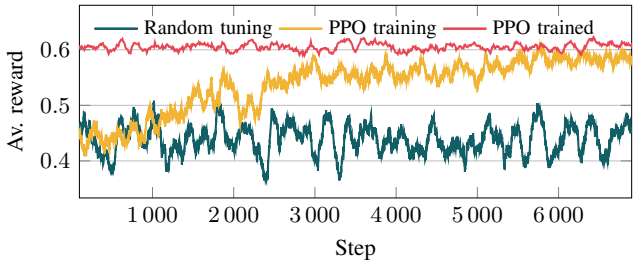


Fig. 3: Comparison of the reward obtained over time.

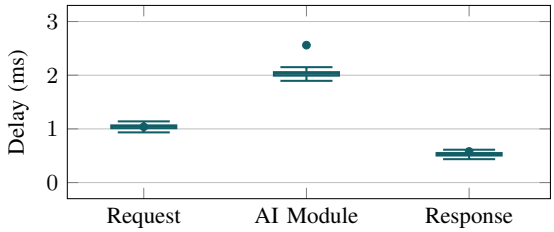


Fig. 4: Latency overhead from the REST API.

embraces the time elapsed since the metric update (from the connector to the module) until the reception of the action to execute (from the module to the connector) as shown in Figure 2a. The total latency is broken down into three key components: i) the time required to perform the HTTP request with state and reward generated by the RAN; ii) the AI module processing time, which includes the DRL agent's decision-making; and iii) the time to send back the HTTP response including the selected action (i.e. a configuration for the scheduler) for being applied by the RAN. The delay in each case is represented using boxplots, where the lower and upper edges of each box correspond to the 25th and 75th percentiles, respectively. The whiskers extend up to 1.5 times the Inter Quartile Range (IQR), approximately covering the range between the 2nd and 98th percentiles. Filled circles within each boxplot denote the average time.

As can be seen, in the tests that we carried out the average time needed to obtain the requested action was approximately 4 ms, while 5G-LENA requires, in average, around 5 seconds of simulation time before a new request could be made. Hence it can be said that the overhead introduced by the API itself, including the processing time induced by the DRL decision-making, does not hinder real-time operation, making the approach suitable for real-world testbeds. Furthermore, the API latency is almost negligible compared to the time required by 5G-LENA to simulate the impact of the selected action on the RAN over the configured 2-second time window. It is worth noting, however, that the processing time of the AI module may vary, depending on the complexity of the DRL model and the computation capabilities of the underlying hardware, which also influences the time required by 5G-LENA to simulate the RAN environment.

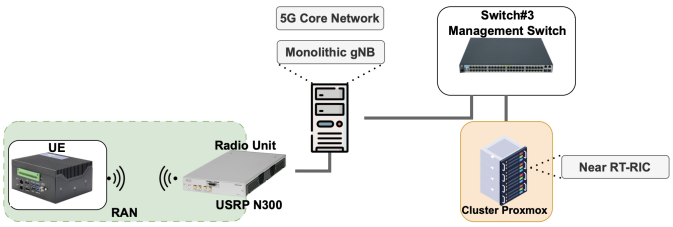


Fig. 5: Bare-Metal Deployment.

V. Future Extensions

This section explores expected future extensions of the proposed OI3A architecture within the O-RAN community and standard. For the purpose of this work, we began setting up a real deployment that involves a small-scale, end-to-end 5G testbed compliant with the O-RAN specifications. The deployment emulates a realistic environment where the interaction between the RAN, core network, and the Near-RTRIC can be thoroughly evaluated.

As illustrated in Fig. 5, the testbed includes all major components of a 5G SA system: a monolithic gNB, a 5G Core, a RU, and a UE, as well as the O-RAN Software Community (OSC) Near-RT RIC running as a virtualized service. We plan to leverage a suite of open-source tools and platforms to build and manage a 5G Standalone (SA) network. Specifically, on the RAN side, we will evaluate two open-source implementations of the 5G gNB: the srsRAN Project [19] and OpenAirInterface5G [20]. On the other side, the 5G Core Network, we plan to adopt the Open5GS implementation [21], which offers a modular and standards-compliant core network for both 4G and 5G.

After this real-world deployment, the proposed OI3A architecture can be thoroughly evaluated using a real testbed. In particular, we will expand OI3A to showcase its compatibility across the O-RAN real testbed. Moreover, we also expect to provide template connectors and AI modules as open-source artifacts to the scientific community, so they can leverage OI3A as part of a different AI development process. Finally, we also expect to implement use cases based on the asynchronous design variant to compare the performance of both approaches.

VI. Conclusion and Future Works

The growing interest in fully integration AI in 5G and 6G networks, especially in those that follow the O-RAN specification, is met with the complexities of coupling AI models and the underlying infrastructure. Moreover, as online training and evaluation becomes a necessity, its operation in real testbeds becomes unfeasible, and leveraging simulation-based approaches requires bridging the gap between the particularities of simulated and real environments. To address these issues, this work proposes OI3A, an architecture to easily integrate AI models into O-RAN, enabling seamless transition from simulated to real testbeds. The evaluation that has been made, which successfully integrates the proposed approach

into ns-3 and 5G-LENA, shows that OI3A enables AI modules, such as DRL modules, to be easily integrated with O-RAN networks with minimal overhead.

In the future, we will expand OI3A to showcase its compatibility across O-RAN simulators, emulators, and real testbeds. Moreover, we also expect to provide template connectors and AI modules as open-source artifacts to the scientific community, so they can leverage OI3A as part of a different AI development process. Finally, we also expect to implement use cases based on the asynchronous design variant, to compare the performance of both approaches.

ACKNOWLEDGMENTS

This work has been funded by the European Union under the MSCA project RENOS (contract number 101205037). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. This work is also supported by the European Commission's Horizon Europe, Smart Networks and Services Joint Undertaking, research and innovation program under grant agreement #101139282, 6G-SENSES project, as well as by the Spanish Government, with the projects PDC2022-133465-I00, PID2021-124054OB-C31, TED2021-130913B-I00 funded by MICIU/AEI/10.13039/50100011033 and the "European Union NextGenerationEU/PRTR", and 6GBLUR/JOINT (TSI-063000-2021-57). It has also received funds from the regional Cantabria and Extremadura Governments through the TCNIC program (2023/TCN/002) and Concepción Arenal program, grant UC-23-40, and the grant GR21133 from the Department of Economy, Science and Digital Agenda of the Government of Extremadura. It was also supported by the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART"). CUP: J33C22 002880001.

REFERENCES

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

[2] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "Open, programmable, and virtualized 5g networks: State-of-the-art and the road ahead," *Computer Networks*, vol. 182, p. 107516, 2020.

[3] J. L. Herrera, S. Montebugnoli, P. Bellavista, and L. Foschini, "Enabling Reusable and Comparable xApps in the Machine Learning-Driven Open RAN," in *2024 IEEE 25th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 2024, pp. 37–42.

[4] H. Yin, P. Liu, K. Liu, L. Cao, L. Zhang, Y. Gao, and X. Hei, "ns3-ai: Fostering artificial intelligence algorithms for networking research," in *Proceedings of the 2020 Workshop on ns-3*, 2020, pp. 57–64.

[5] P. Gawłowicz and A. Zubow, "ns3-gym: Extending openai gym for networking research," 2018. [Online]. Available: https://arxiv.org/abs/1810.03943

[6] S. Bi, L. Huang, H. Wang, and Y.-J. A. Zhang, "Lyapunov-Guided Deep Reinforcement Learning for Stable Online Computation Offloading in Mobile-Edge Computing Networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 11, pp. 7519–7537, 2021.

[7] I.-S. Comşa, S. Zhang, M. E. Aydin, P. Kuonen, Y. Lu, R. Trestian, and G. Ghinea, "Towards 5G: A Reinforcement Learning-Based Scheduling Solution for Data Traffic Management," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1661–1675, 2018.

[8] R. Li, Z. Zhao, Q. Sun, C.-L. I, C. Yang, X. Chen, M. Zhao, and H. Zhang, "Deep Reinforcement Learning for Resource Management in Network Slicing," *IEEE Access*, vol. 6, pp. 74 429–74 441, 2018.

[9] S. Imtiaz, H. Ghauch, G. P. Koudouridis, and J. Gross, "Random forests resource allocation for 5G systems: Performance and robustness study," in *IEEE WCNCW 2018*, 2018, pp. 326–331.

[10] A. Chilmulwar and V. Sinha, "A Novel Machine Learning Based MAC Scheduler Algorithm using ARIMA," in *IEEE CCNC 2019*, 2019, pp. 1–8.

[11] X. Guo, Z. Li, P. Liu, R. Yan, Y. Han, X. Hei, and G. Zhong, "A Novel User Selection Massive MIMO Scheduling Algorithm via Real Time DDPG," in *IEEE GLOBECOM 2020*, 2020, pp. 1–6.

[12] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "ColO-RAN: Developing machine learning-based xApps for open RAN closed-loop control on programmable experimental platforms," *IEEE Transactions on Mobile Computing*, vol. 22, no. 10, pp. 5787–5800, 2022.

[13] N. Patriciello, S. Lagen, B. Bojovic, and L. Giupponi, "An E2E simulator for 5G NR networks," *Simulation Modelling Practice and Theory*, vol. 96, p. 101933, 2019. [Online]. Available: https://doi.org/10.1016/j.simpat.2019.101933

[14] Mathworks, "5G Toolbox - MATLAB," 2021. [Online]. Available: https://mathworks.com/products/5g.html

[15] OpenAPI Initiative, "OpenAPI Specification Version 3.1.1," 2024. [Online]. Available: https://swagger.io/specification/

[16] S. Schwichtenberg, C. Gerth, and G. Engels, "From Open API to Semantic Specifications and Code Adapters," in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 484–491.

[17] M. Jones, Microsoft, J. Bradley, P. Identity, N. Sakimura, and NRI, "JSON Web Token," RFC 7519, May 2015. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7519

[18] N. Villegas, A. Larrañaga, L. Diez, K. Koutlia, S. Lagén, and R. Agüero, "Extending QoS-aware scheduling in ns-3 5G-LENA: A Lyapunov based solution," in *Proceedings of the 2024 Workshop on Ns-3*, ser. WNS3 '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 54–59.

[19] srsRAN Project, 2025. [Online]. Available: https://www.srslte.com/5g

[20] OpenAirInterface, 2025. [Online]. Available: https://gitlab.eurecom.fr/oai/openairinterface5g

[21] Open5GS, 2025. [Online]. Available: https://open5gs.org/open5gs/