A Tutorial on O-RAN deployment solutions for 5G: From Simulation to Emulated and Real Testbeds

Juan Luis Herrera, Sofia Montebugnoli, *Student Member, IEEE,* Domenico Scotece, *Member, IEEE*
Luca Foschini, *Senior Member, IEEE,* Paolo Bellavista *Senior Member, IEEE,*

***Abstract*—The Open Radio Access Network (O-RAN), manifested through the specifications established by the O-RAN Alliance, stands ready to transform the telecommunications ecosystem. In particular, the O-RAN Alliance reference architecture proposes hierarchical structures supported by several interfaces, including non-real-time and near-real-time functions. Non-real-time functions, denoted as rApps, encompass services, configurations, policy management, and RAN analytics. Conversely, near-real-time applications, denoted as xApps, serve as an open compute edge platform hosting applications from multiple vendors. Therefore, understanding the testing and deployment processes of rApps and xApps is pivotal for both researchers and practitioners. In this tutorial, we evaluate different 5G RAN deployments to determine the state of the art of available open-source solutions and assess their readiness for real cloud continuum environments by targeting solutions for simulation, emulation, and real testbeds.**

This tutorial empowers developers to gain a comprehensive understanding of O-RAN technologies, their development, and operations, enabling them to make well-informed choices for their target technologies. It considers both qualitative and quantitative benchmarks, as well as use cases, and facilitates a seamless transition of applications from simulated environments to real-world scenarios. Moreover, this tutorial provides a hands-on guide on setting up and migrating an xApp from a simulated environment to a real testbed. It includes a guide on deploying Service Management, Orchestration, and Non-Real-Time RAN Intelligent Controller and rApp. Finally, we identify existing open challenges and trace future directions to enhance the availability, portability, reliability, and configurability of 5G RAN deployments.

***Index Terms*—Open RAN, O-RAN, xApps, DevOps, Cloud Continuum, Simulator, Emulator, 5G**

I. Introduction

TRADITIONALLY, Radio Access Networks (RANs) have been implemented using a *black-box* approach, in which hardware, software, and interfaces were proprietary [1]. While vendors usually provide tools and software that can interact with their own RAN elements, the use of proprietary interfaces for communication complicates the interaction across RAN elements from different vendors, thus leading to vendor lock-in [1]. This vendor lock-in further leads to limitations in the control operators can have over the RAN, as functionalities that are not provided by the vendor's software may be complex or impossible to use, even if they are supported by the underlying hardware [1]. To address the limitations of black-box systems, the Open Radio Access Network (O-RAN) specification arises as an architectural paradigm that proposes a non-proprietary version of the RAN [2]. A RAN that follows the O-RAN specification can still use proprietary hardware and software, but its elements must communicate through standard and open interfaces that ensure interoperability across vendors [2]. The O-RAN architecture supports both Fourth generation of mobile networks (4G) and Fifth generation of mobile networks (5G) and facilitates interoperability in scenarios where both network types are used [1], [2].

From the control plane perspective, O-RAN proposes to differentiate between three timescales [1]: Real-Time, Near-Real-Time, and Non-Real-Time. This differentiation has its basis in the different requirements exhibited by each timescale. To perform control operations, O-RAN defines the RAN Intelligent Controller (RIC), which includes the software elements that communicate with the RAN to obtain information and send control commands [3]. Due to the timescale differentiation, O-RAN specifies two different RIC, the Near Real-Time RAN Intelligent Controller (Near-RT RIC), and the Non Real-Time RAN Intelligent Controller (Non-RT RIC), each concerned with control at its respective timescale. While Real Time (RT) requirements are so strict that they must be embedded [3]. A key feature of RICs is that they enable developers to create *services* or *applications* that can interact with the RAN. To introduce the programmability of the RAN, as well as perform RAN control with customized algorithms [4]. RAN applications aimed at the near-RT timescale are known as Extended Application (xApp), while those for the non-RT timescale are deemed RAN Intelligent Controller Application (rApp).

As O-RAN enforces the use of open interfaces for communication, xApps and rApps are regular software components that interact with the RAN. Thus, developers of traditional software applications can apply their knowledge to the development of xApps and rApps, including efficient implementations of control algorithms and Development and Operations (DevOps) techniques to enable changes in the code to be automatically tested and deployed, minimizing the delivery

J.L. Herrera, S. Montebugnoli, D. Scotece, L. Foschini, and P. Bellavista are with the Department of Computer Science Engineering, University of Bologna, Italy. (e-mail: [juanluis.herrera, sofia.montebugnoli3, domenico.scotece, luca.foschini, paolo.bellavista]@unibo.it).

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART") CUP: J33C22002880001

Manuscript received June 12th, 0000; revised June 12th, 0000.

time of new or improved functionality [5]. However, these techniques require new versions of xApps and rApps to be tested not only in a traditional *functional* manner (e.g. unit and integration tests), but it is also key to test their *performance* to ensure the new versions do not worsen the performance of the RAN and meet their response time requirements. Furthermore, it is crucial to perform these tests in multiple scenarios to minimize the difference between test and real performance. In this context, O-RAN simulation tools can allow xApps and rApps to be tested in realistic scenarios, enabling the comparison of performance across different versions or even different xApps/rApps with the same objective (e.g. Machine Learning (ML)/Artificial Intelligence (AI)-based xApps that use different models). However, developers may find difficulties when their applications move from simulated to emulated or real environments, requiring changes to accommodate the differences, making it crucial to navigate the path to convert prototypes tested in simulations to real, industry-ready software. Practical deployments that include all the elements in the O-RAN architecture often find compatibility, data formatting, interpretation, availability, or orchestration issues, highlighting the need to carefully select appropriate technologies at all stages of development.

This tutorial aims to support current and future O-RAN developers in prototyping O-RAN control plane software and testing it in both simulated and emulated/real testbed scenarios. We also aim to provide a comprehensive view of the adoption of DevOps tools by the different projects to assess the ease of use of these solutions for the developers. This tutorial reviews enabling technologies for O-RAN simulation, emulation, real testbed deployment, and software development for the RICs. To the best of our knowledge, it is the first tutorial presenting a full deployment of the RAN that includes all its functions: Open Radio Unit (O-RU), Open Distributed Unit (O-DU), Open Centralized Unit (O-CU), Near-RT RIC, Non-RT RIC, and Service Management and Orchestration (SMO) in simulated, emulated and real testbeds. The tutorial evaluates the compatibility and interplay of the various open-source projects, aiding readers in understanding the O-RAN architecture and its development processes. Also for this reason, the tutorial considers only open-source tools available to the entire community to promote the openness of RAN, research collaboration, and innovation, enabled by the freedom and flexibility of leveraging these projects. Various options for developing and testing O-RAN software are examined, highlighting their compatibility and pros and cons for informed decision-making. Additionally, gaps in current O-RAN implementations are identified, suggesting future research and development directions. In brief, the main objectives and novel aspects of this tutorial are as follows:

- Providing readers with the necessary background in O-RAN architecture, network functions, functional splits, differentiation between data and control planes, and the interfaces that govern their interactions. As well as an introduction to the DevOps approach to understand the readiness of the O-RAN projects.
- Analyzing the state-of-the-art in terms of O-RAN simulation, emulation, and real testbed deployment, including the parts of the O-RAN architecture they simulate, emulate or implement, the features they support, their compliance with specifications, and the target use cases for each.
- Comparing the existing projects that implement O-RAN control plane components, such as Near-RT RICs, Non-RT RICs, and SMO, in terms of their architectures, features, Application Programming Interface (API), elements, platforms, and their conceptual differences.
- Assessing the interoperability across O-RAN control plane entity implementations, simulators, and real testbed and emulation software, thus allowing readers to choose a version of these technologies that comply with their use case, as well as to learn about the idiosyncratic differences that exist in moving from simulated to real testbeds based on the chosen technologies.
- Presenting the readers with a detailed hands-on tutorial on the setup and migration of an xApp from a simulated environment to a real testbed to show the viability of the process and serve as a starting point for their own development and testing.
- Presenting to the best of our knowledge the first tutorial on the deployment of a Non-RT RIC and SMO in a Kubernetes cluster and Docker with simulated interfaces.
- Discussing the gaps in the state-of-the-art of the O-RAN technologies, therefore highlighting the current needs in terms of research and implementation of the O-RAN architecture and pointing towards future directions.

The remainder of this tutorial is shown in Fig. 1 and is organized as follows. Sec. II reviews related surveys and tutorials on O-RAN and highlights the need for the current work. Sec. III provides the reader with essential knowledge of the O-RAN architecture, covering perspectives from the user plane and control plane in their deployment aspects. Sec. IV discusses and compares the available O-RAN simulation tools, while Sec. V outlines the projects focused on emulated and real testbed implementations. Sec. VI details the various O-RAN control plane components and examines their compatibility with existing simulation and emulation/real testbed tools. Sec. VII offers a hands-on demonstration of migrating O-RAN control plane software from a simulated environment to a real testbed. Sec. VIII addresses current discussion gaps and open challenges in the simulation and implementation of O-RAN user and control planes. Finally, Sec. IX suggests future research direction, and Sec. X concludes the work outlining its main contributions.

II. Related works

This section provides an in-depth examination of surveys and tutorials related to state-of-the-art O-RAN development and deployment solutions. Notably, it does not aim to cover the entirety of the state-of-the-art in O-RAN but specifically focuses on reviewing surveys and tutorials in this domain. As detailed in the following sections, the tutorial distinguishes itself by providing a thorough overview of O-RAN deployment methods, including simulation, emulation, and real-world

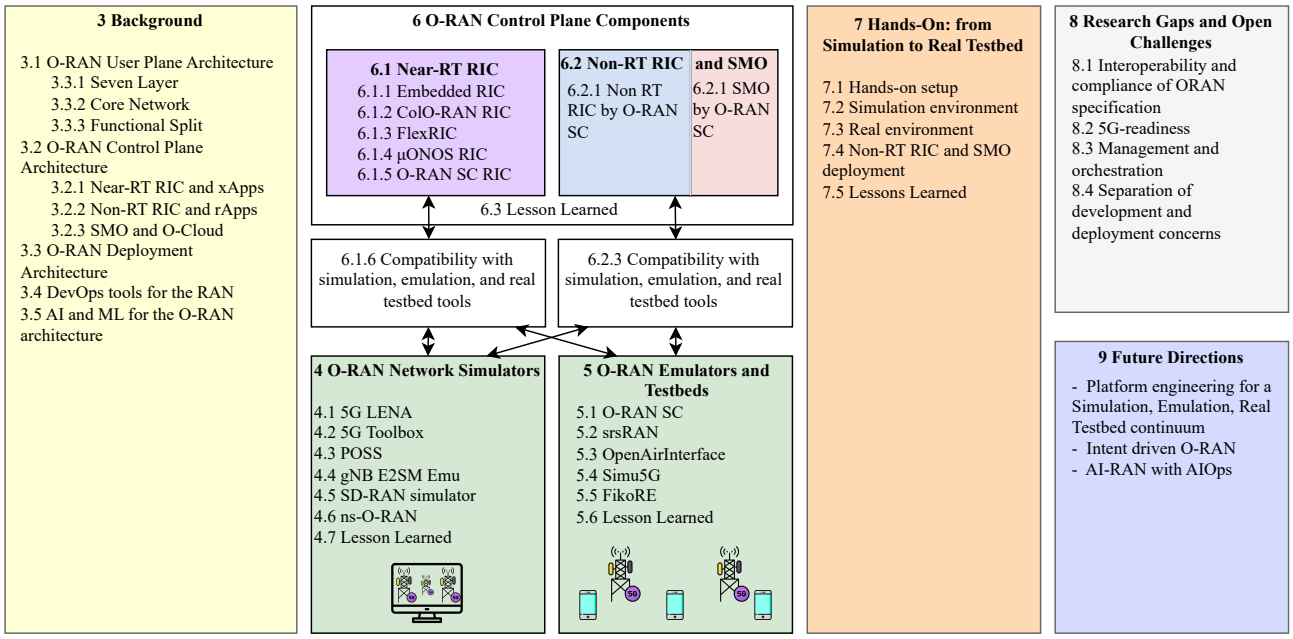


Fig. 1: Detailed paper structure depicting the main sections of the paper and their organization

TABLE I: Summary of Recent O-RAN Surveys and Tutorials

Year	Refs	Tutorial	Next-Gen 5G	Architecture	Control Plane	Simulation	Emulation	Testbed
2016	Agiwal *et al* [6]		√	√				
2016	Peng *et al* [7]		√	√				
2019	Hossain *et al* [8]		√	√				
2020	Gkonis *et al* [9]		√			√		
2022	Pana *et al* [10]		√	√				
2022	Arnaz *et al* [11]		√	√	√			
2023	Shen *et al* [12]		√	√				
2023	Hoffmann *et al* [5]		√	√	√			
2023	Upadhyaya *et al* [13]		√	√	√			√
2023	Aryal *et al* [14]		√	√	√			
2023	Amini *et al* [15]		√	√	√			
2024	Azariah *et al* [16]		√	√	√			
2024	Chen *et al* [17]		√	√				
2024	Polese *et al* [18]		√	√	√			
2024	Marinova *et al* [19]		√	√	√			
2024	Ngo *et al* [20]	√		√	√			√
2023	Polese *et al* [1]	√	√	√	√	√		
2023	Silva *et al* [21]	√		√	√		√	
2024	Santos *et al* [22]	√			√			√
	This Work	√	√	√	√	√	√	√

testbeds. Additionally, it features a catalog of O-RAN control plane solutions, their compatibility with deployment tools, and step-by-step instructions for developing xApps/rApps.

We begin reviewing the state-of-the-art surveys not so closely related to this tutorial, such as those on next-gen 5G networks, and progress to studies focused on O-RAN architectural design and O-RAN control plane. In particular, we provide a comparison of existing surveys and tutorials from multiple perspectives, including next-generation 5G networks, 5G architecture, the control plane, and various evaluation methods such as simulation, emulation, and real testbeds. Finally, we conclude the state-of-the-art analysis with the latest tutorial proposals in the field of O-RAN deployment.

Most of the literature covers studies on next-generation 5G networks and RAN design methodologies, which are not fully focused on control plane components and evaluation. For example, early works published a few years ago, [6]–[8] explored future RAN designs for next-generation 5G networks, focusing on architectures, key techniques, and open issues. Along similar lines, the study by Pana et al. [10] examines 5G RAN architectures, focusing on efficiency, spectrum, energy, latency, and resource management. It highlights current issues and existing solutions. On the contrary, Shen et al. [12] investigate the most important key research areas for next-generation networks and propose multi-component Pareto optimization for future Sixth generation of mobile networks (6G) advancements. Finally, most recently, the survey proposed in [17] examines the progression of RAN, introduces a fully decoupled RAN for 6G, discusses enabling technologies, identifies open challenges, and seeks to inspire further research aimed at enhancing future performance.

Among the O-RAN initiatives, recent works have begun exploring control plane components that enable the integration

and programmability of the O-RAN architecture. In particular, the work in [11] explores O-RAN vision for vendor-agnostic, AI-integrated RAN solutions, focusing on new 5G and 6G use cases. Similarly, the work in [5] examines O-RAN opportunities and challenges. It addresses third-party application development issues, compares open platforms for xApps, and highlights implementation challenges and solutions for various applications, advocating for joint cooperation to advance O-RAN. Furthermore, Upadhyaya et al. [13] explore O-RAN significance in future cellular networks, focusing on its openness and intelligence to tackle 5G challenges. It introduces a framework for AI-based RAN management. The testbed capabilities, challenges within O-RAN, and a use case on Key Performance Measurement (KPM) are discussed. A more detailed review of the architecture evolution is presented in [14], which surveys the potential of O-RAN to disrupt traditional networks by enabling disaggregation of hardware and software, fostering a competitive ecosystem. Challenges like interoperability and AI/ML management are also discussed alongside existing solutions. Most of the mentioned works lack a thorough evaluation component, such as simulation, emulation, or real-world experiments. Therefore, the survey in [9] covers advanced simulation techniques for 5G networks, focusing on performance metrics. It addresses issues with directional beams, coexistence with 4G, and potential applications in smart grids, emphasizing the need for comprehensive simulation environments.

More focused on practical implementation, the work proposed in [15] reviews the integration of AI/ML in O-RAN control plane, addressing the scarcity of publicly available data and researchers' efforts to collect data independently. Another survey that explores solutions from an open-source perspective to enhance flexibility, performance, and cost-efficiency in O-RAN deployment and operation is presented in [16]. In particular, it is focused on O-RAN evolution, technologies, projects, standardization efforts, challenges, and future research directions. Similarly, the work proposed in [19] provides insights and guidelines for leveraging O-RAN's open, softwarized, and intelligent framework, focusing on RICs and AI/ML-driven network optimization, along with applications like xApps and rApps for advanced service types such as V2X and Industry 5.0. Finally, the work proposed by Ngo et al [20] explores O-RAN for 5G, highlighting multi-vendor compatibility, the central role of RIC, and the impact of open-source projects. It details the implementation and evaluation of RIC applications on a commercial-grade 5G network.

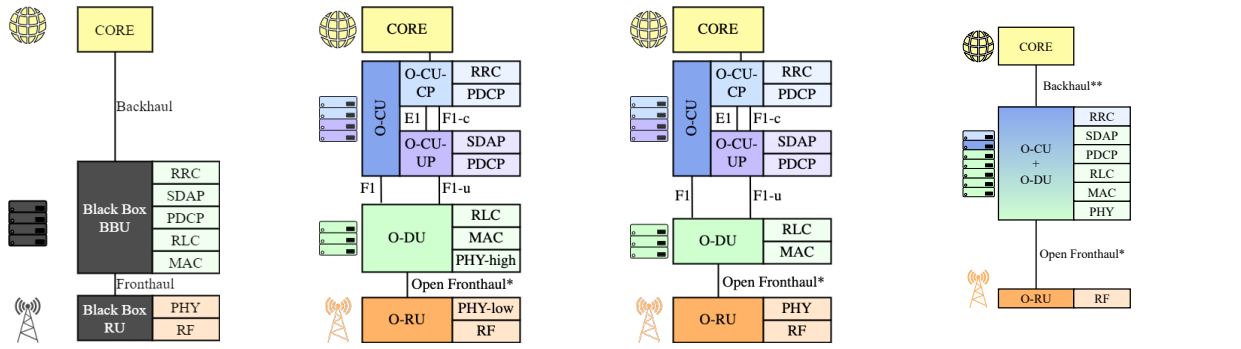
Regarding the evolution of 6G networks, Polese et al. [18] examine the transformative potential of O-RAN for 6G, with a focus on disaggregated architecture, cloudification, AI-driven control, and the necessity for continued research, development, and standardization efforts.

A major area of research focuses on tutorials related to the implementation of O-RAN. The tutorial proposed by Polese et al. [1] reviews O-RAN revolutionary impact on telecom, detailing its architecture, interfaces, intelligent controllers, AI/ML integration, security, standardization challenges, research platforms, and future development directions. Unlike our proposed tutorial, that tutorial does not emphasize deployment and development solutions for xApps and rApps. Specifically, it does not examine existing open-source solutions for simulation, emulation, and real testbeds, which are essential for effectively evaluating the potential of the O-RAN framework. Furthermore, the work in [21], proposes an orchestration framework for 5G RAN using Open Air Interface (OAI) solution compliant with O-RAN standards. The proposed architecture was validated with an experimental prototype, showing interoperability between RAN functions and the RIC, while comparing the status of existing OSS solutions for Near-RT RIC deployment. Finally, the work presented in [22] investigates the design and configuration of xApps. Specifically, it offers best practices to support xApps developers in testing their applications and provides a testbed evaluation for xApps.

Unlike the previously discussed works, this tutorial distinguishes itself by providing comprehensive solutions for the development and deployment of xApps and rApps within the O-RAN architecture framework. In recent years, the industry and research community have been tirelessly working on various aspects of O-RAN. For instance, significant efforts have focused on its standardization and role in shaping future networks [23], [24]. The O-RAN literature encompasses a range of topics, including the application of ML techniques [25]–[28], and various architectural perspectives [29], [30]. Furthermore, while the infrastructure is evolving toward 6G Radio Access Networks, the literature has explored Digital Twin (DT) technology and intelligence within O-RAN [31], [32]. On the contrary, the development of xApps/rApps, being one of the most important enablers for the next-generation RAN networks, and the surveys considering their applications and future directions are available [5], [22], [33]. In particular, to the best of our knowledge, there are only a couple of papers published in the last year that explore O-RAN and RIC [21], [34]. Moreover, the proposed tutorial covers next-generation 5G networks, RAN design methodologies, and O-RAN control plane components. Specifically, it compares various existing solutions for Near-RT RICs, Non-RT RICs, and SMOs, as well as network simulators and emulators. To the best of the authors' knowledge, it is the first work to examine the complete end-to-end DevOps cycle of xApps/rApps, encompassing creation, simulation, and real-world deployment. Finally, this tutorial provides a set of guidelines for readers to set up and test xApps from simulations to real testbeds.

A comparison of this work with existing surveys and tutorials in the literature is summarized in Table I. In this table, we propose a multi-aspect comparison of the existing surveys and tutorials where we highlight the content of each reference in terms of the main key topics addressed and types of analyses conducted. The state-of-the-art analysis reveals that most surveys focus on next-generation 5G networks, the O-RAN architecture, and the control plane, with only a few incorporating simulation or testbed analysis. In contrast, tutorials in this field lack coverage of certain aspects of the O-RAN architecture, and none encompass at the same time simulation, emulation, and testbeds.

To the best of the authors' knowledge, this tutorial is the first to comprehensively cover all aspects of the O-RAN architec-



(a) *Option 1* functional split (b) *Option 7.2x* functional split (c) *Option 6* functional split (d) *Option 8* functional split

Fig. 2: O-RAN user plane architecture and functional splits compared to the traditional approach. An asterisk marks that Open Fronthaul is not a standardized interface, while two asterisks mark that Backhaul is not standardized either and is a different interface from Open Fronthaul.

ture including the control plane analysis while providing step-by-step guidelines for developing and deploying xApps/rApps in simulated, emulated, and real testbed environments.

III. Background

This section offers the essential background on O-RAN to provide the reader with the necessary knowledge regarding the architecture, network functions, control and user planes, and system compatibility. It also focuses on the development and deployment of the RAN control applications, known as xApp and rApp, at the edge. At the end, a general approach to DevOps tools in the RAN is also presented.

A. O-RAN user plane architecture

Traditionally, networks are split into two planes: the *user plane* or *data plane*, which is concerned with the transmission and communication of data, and the *control plane*, which monitors and manages how the user plane performs its tasks [35]. In the case of RANs, the user plane is concerned with the transmission of data through radio frequency signals, which includes, among other tasks, transmitting and receiving these signals from the user's devices to the RAN, encoding and decoding the information in the signals, configuring how the available bandwidth is to be shared across connected devices, controlling the device-to-RAN connections, providing the resources the RAN allocates to the various users, or the connection to the core network (e.g., the Internet). Moreover, control messages negotiated by the control plane are also eventually exchanged at the user plane, which is also concerned with their transmission. All these functions are realized and structured through 7 layers, defined by 3GPP 5G-NR RAN2 [36] and in [37] to move from a monolithic architecture to a disaggregated one, each assigned to a specific task within the user plane, and a set of functions known as *core*, that are detailed in the remainder of this subsection.

1) Seven Layers:

a) RF layer: The lowest layer of the user plane is the Radio Frequency (RF) layer. This layer is below the classic physical layer, as its tasks are not directly related to data transmission. Instead, the role of the RF layer is to establish and maintain a high-quality medium between the User Equipment (UE), and the rest of the RAN [35].

In other words, RF layer can be compared to the manufacturing and maintenance of the network links (e.g., optic fiber cables) in traditional wired networks.

b) PHY layer: on top of the RF layer, there is the Physical (PHY) layer, which is concerned with transforming information into signals and transmitting them [35]. This process involves multiple functions: 1) Digital-to-Analogue signal conversion, 2) Analogue-to-Digital signal conversion, 3) beamforming, 4) a fast Fourier Transform (FFT), 5) Cyclic Prefix management, 6) precoding, 7) Resource Element mapping into subcarriers and Orthogonal Frequency-Division Multiplexing (OFDM) symbols, 8) RF layer mapping, 9) modulation, and 10) scrambling. These functions are not explored in detail as they are out of the scope of the tutorial. Due to the vast amount of functions performed by the PHY layer, it is possible to split it into two sub-layers [35]: Physical-low (PHY-low), which contains the first five functions, which are the time-domain tasks, and Physical-high (PHY-high), which contains the rest of the functions that belong in the frequency domain.

c) MAC layer: On top of the PHY layer, we find the Medium Access Control (MAC) layer, which controls how each of the UE can use the shared resources of the RAN [35]. The MAC layer provides crucial functions to the upper layers in the protocol stack, allocating resources and scheduling the transmissions to and from each UE, managing error corrections and retransmissions or multiplexing and de-multiplexing traffic into transport blocks for efficiency [35]. One key function in the MAC layer is *Logical Channels*, which allows the separation of user plane messages (which use Data Radio Bearer (DRB)) and control plane messages (sent and received through Signal Radio Bearer (SRB)) [35].

d) RLC layer: The Radio Link Control (RLC) allows messages to be sent through the logical channels provided by the MAC layer in three different modes: acknowledged mode, unacknowledged mode, and transparent mode [35]. Transparent mode offers no modifications to the data and

hence, no overhead, while unacknowledged mode provides segmenting and concatenation, at the cost of a small overhead. Acknowledged mode offers features from unacknowledged mode, adding functions to ensure correct data transmission and flow control and larger overhead.

e) PDCP layer: On top of the RLC layer, the Packet Data Convergence Protocol (PDCP) verifies and transforms packets (e.g., IP packets) into data that can be sent and received from the RAN, optimizes their size, routes them, and provides confidentiality [35]. PDCP provides sequence numbering, ciphering, and routing to all the packets. Moreover, PDCP offers header compression to user plane packets, and integrity protection and verification to control plane ones.

f) SDAP layer: The Service Data Adaptation Protocol (SDAP) is responsible for mapping the different packet flows from the network into their corresponding Quality of Service (QoS) flows, where a single QoS flow may contain messages from different sources but with similar QoS requirements [35]. Once this mapping is performed, SDAP must also map each QoS flow to a given DRB from the lower layers, thus sending the data with the required QoS. However, SDAP only acts with user plane traffic.

g) RRC layer: The Radio Resource Control (RRC) layer handles the control plane traffic. Traditionally, especially in low-end systems, the configuration of the lower layers was fixed. However, it is instead desirable to be able to dynamically change control parameters at the different layers of the user plane to allow the RAN to adapt to different situations. The RRC layer is precisely the one responsible for coordinating and configuring the rest of the layers using control messages.

2) Functional splits: To implement this layered architecture, it is necessary to decide which device or devices will host the software implementing each of the layers. This strategic decision depends on the specific deployment: while the RF and the PHY-low layers are strongly coupled with the physical radio hardware, the rest of the layers can be implemented and deployed remotely into more general computing hardware. This has led to various proposals on how many computing elements to have and how to distribute the different layers across them, namely *functional splits*. Fig. 2 presents the four main functional splits in O-RAN networks [38], as better detailed in the following paragraphs.

The traditional functional split, known as *option 1*, is the one that has been mainly used by black-box, non-O-RAN networks and is depicted in Fig. 2a. In option 1, the RF and PHY layers are directly implemented into the radio hardware, integrating what is known as the O-RU. In traditional RAN, this Radio Unit (RU) is implemented as a black box piece of hardware, that provides a proprietary implementation of the PHY and RF layers. The other 5 layers, from MAC to RRC, are implemented in a computing device known as the Baseband Unit (BBU), which also provides proprietary implementations for them. The O-RU and BBU communicate through an interface known as the *Open Fronthaul*, and is traditionally also proprietary, limiting interoperability across vendors. On the other hand, the interface that connects the BBU to the core is the *Backhaul*, and may also be proprietary. Furthermore, the O-RU and the BBU can be provided as a single hardware box or *standalone 5G Next Generation Node B (gNB). In both standalone and disaggregated gNB, uplink user plane data is sent from UE to the O-RU, transmitted to the BBU through the Open Fronthaul, and reaches the core through the Backhaul*, while downlink data takes the reverse optipath. Finally, it is noteworthy that O-RAN evolves the option 1 functional split, as it is considered an older functional split.

The next functional split, depicted in Fig. 2b, is the *option 7.2x functional split*. This is the most accepted and flexible functional split in the O-RAN specification, although not the only one. The option 7.2x functional split also has an O-RU, that implements the RF and PHY-low layers, posting the functions that are more coupled to the radio hardware. The PHY-high layer, along with MAC and RLC layers, are provided by a different device: the O-DU, which communicates with the O-RU through an interface known as *Open Fronthaul*, which contains an asterisk in the figure to remark that it is not yet standardized. Nonetheless, it is key that the Open Fronthaul interface is open and allows interoperability across vendors and open-source implementations. Moving on with the three topmost layers, PDCP, SDAP, and RRC, they are implemented in the O-CU that communicates with the O-DU through the standardized F1 interface. The option 7.2x functional split further divides the O-CU into two: the Open Central Unit User Plane (O-CU-UP), implementing PDCP and SDAP, and the Open Central Unit Control Plane (O-CU-CP), for PDCP and RRC. This split allows operators to disaggregate the gNB in 4 separate devices (O-RU, O-DU, O-CU-UP, O-CU-CP). If this option is taken, the O-DU communicates with the O-CU-UP through the F1-u interface, and with the O-CU-CP through the F1-c interface. Moreover, the O-CU-UP and the O-CU-CP can communicate through the E1 interface. Finally, the O-CU connects to the core through a non-standard Backhaul.

Another alternative is the *option 6 functional split*, as shown in Fig. 2c. This functional split option is largely similar to the 7.2x split although the mapping across layers changes slightly, as functional split option 6 implements the complete PHY layer at the O-RU, leaving to the O-DU only the MAC and RLC layer. The O-RU and O-DU communicate through an Open Fronthaul, which may be different from the Open Fronthaul interface used in other functional splits, as it is not standard.

The other functional split that is currently being studied for O-RAN is the *option 8 functional split*, depicted in Fig. 2d, and defined by 3GPP in [37]. This functional split maps exclusively the RF layer to the O-RU and implements the rest of the layers into the O-DU(+O-CU). Split 8 is simple but limits the disaggregation across layers and elements in the RAN.

To enable interaction among the various components, the 5G Femto Application Platform Interface (FAPI) published by the Small Cell Forum (SCF) [39] is a suite of specifications that enable small cells to be built up piece-by-piece using components from different suppliers. This is further enriched by the network Functional Application Platform Interface (nFAPI) specification that wraps these APIs to make them transportable over network connections, making nFAPI an

external network interface between O-RU and O-DU network nodes.

3) Core: The Core Network (CN) is a fundamental part in 3rd Generation Partnership Project (3GPP) cellular networks, responsible for central management and control functions, as well as routing and forwarding data across the network. It acts as the backbone of mobile communication systems, connecting various RAN to external networks, such as the internet or other service networks. The system architecture for 5G Core (5GC) has been established in the 3GPP standard Release 15, which introduced the Service Based Architecture (SBA) [40]. Rather than relying on monolithic elements, 5G adopts Network Functions (NFs) that modularizes the tasks of the core. Contrary to Evolved Packet Core (EPC) (the 4G Long-Term Evolution (LTE) network), a pivotal aspect of SBA modularization is softwarization and cloudification. For details on how to practically simulate or emulate both the seven RAN layers and the CN, we refer the reader to Section VII.

B. O-RAN control plane architecture

The control plane configures the user plane to deliver quality attributes.

Unlike the user plane, data traffic does not flow through the control plane. Instead, the control plane is dedicated to traffic such as signaling from the RAN, RAN reports on metrics, and commands towards the RAN to configure its operation, encompassed as *control traffic*.

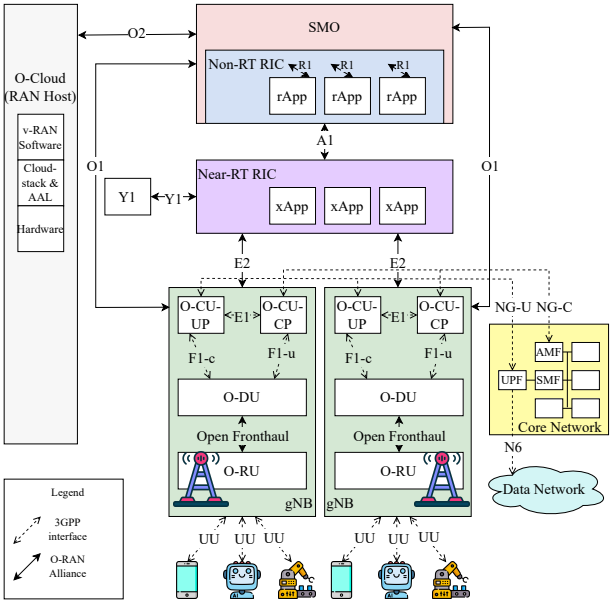


Fig. 3: O-RAN control plane architecture, using the 7.2x split.

Figure 3 shows the O-RAN control plane architecture using the 7.2x split. O-RAN separates the control plane into three timescales, defined by the O-RAN Alliance [2]. Each timescale has certain requirements for the response time of its control actions. First, the RT refers to the instantaneous actions that the control plane must take almost continuously to maintain an operative RAN, responding in $t < 10ms$. Second, the Near-Real Time (Near-RT) timescale is more appropriate for short-term adaptations of the RAN's operation, responding within a window of $10ms \leq t < 1s$. Finally, the Non-Real Time (Non-RT) timescale enables long-term analysis and adaptation of the RAN, with a very loose response time requirement $t \geq 1s$. The element concerned with the RT timescale is the O-DU. However, the O-RAN Alliance is currently discussing to allow custom control logic software or *Operator-defined Real Time Applications (dApps)* to run in the O-DU [41].

For the other two timescales, O-RAN defines the RIC. There are two different RICs, the Near-RT RIC depicted in violet in Fig. 3, and the Non-RT RIC depicted in light blue in Fig. 3, each functioning on its respective timescale. Additionally, they contain different modules, communicate via separate interfaces, serve unique roles, and provide distinct functions. In the Near-RT RIC, these functions are called *xApp*, while in the Non-RT RIC, they are referred to as *rApp*. Finally, O-RAN defines a separate element named SMO, depicted in pink in Fig. 3, to manage the infrastructure as a whole, including RICs, xApp, and rApp. Nonetheless, the specification sets no concrete boundary between the SMO and the Non-RT RIC. As dApp are still under discussion, the remainder of this section focuses on the Near-RT RIC, the Non-RT RIC, and the SMO.

1) Near-RT RIC and xApp: The first element we analyze is the Near-RT RIC, which performs the control tasks at a near-RT scale. Two key concepts in the Near-RT RIC are Key Performance Indicator (KPI) and KPM. KPIs refers to RAN metrics that the operators can be interested in (e.g., Signal to Interference-plus-Noise Ratio (SINR), buffer occupancy, volume of PDCP packets transmitted), while KPMs refers to specific values for these KPIs, (e.g., 39 dB SINR, 30 KB buffer occupancy, or 800 PDCP packets). The Near-RT RIC acts in a loop, obtaining the KPMs for the KPIs used for the tasks it must perform, making control decisions in consequence, and transmitting appropriate control commands back to the RAN, all within the near-RT timescale. To do so, the Near-RT RIC can subscribe to certain KPIs in the RAN to obtain their KPMs, both in a time-based or event-based manner. The communications across the Near-RT RIC and the RAN are performed through the standard E2 interface, connected to the E2 terminations of O-DU, O-CU-UP, and O-CU-CP, as shown in Fig. 3, or a single E2 termination for standalone gNB (in the case of option 8 functional split).

Let us note that the subscriptions to KPIs, the reception of KPM reports, and the generation of control messages in response are not the direct responsibility of the Near-RT RIC. Instead, the Near-RT RIC serves as a way to host *xApp*, which are microservices (i.e., small application components that perform very cohesive operations) that interact with the RAN to perform specific control functions, such as slicing, handover, and retransmission monitoring. Each xApp can subscribe to different KPIs and take different control actions. The O-RAN Alliance also specifies that the Near-RT RIC must host a Shared Data Layer (SDL), which is a data storage system that allows all xApps to share data across themselves and with the Near-RT RIC, effectively serving as both a communication system across xApp and the Near-RT RIC, and a common

storage system.

There are three interpretations of how the Near-RT RIC hosts xApps: as separate components, as integrated components, or as embedded modules. The first interpretation considers that the Near-RT RIC is a set of one or more software components that provide common services to xApps, such as the E2 termination and the SDL. In this case, *separate components* host xApps as natively independent software artifacts that must connect to the Near-RT RIC but can be deployed on different machines. Some RIC implementations follow this interpretation [25], [42], [43]. In the case of *integrated components*, held by other RIC implementations [44], the interpretation is similar, but the Near-RT RIC encompasses a single software component where all its modules, xApps included, are integrated. In this interpretation, a Near-RT RIC can be specialized for certain tasks by adding or removing xApps from its codebase, but must be deployed as a single component. Finally, the *embedded modules* interpretation considers that there is no single Near-RT RIC, and instead, each individual xApp, seen as an independent software artifact, must implement all its functionality. This interpretation thus sees the Near-RT RIC as a specification of the features every xApp must implement [45].

To effectively communicate with the RAN, xApps must use the same format the RAN uses to send KPMs, as well as to receive control commands. These formats are called Service Model (SM), also receiving the name of E2 Service Model (E2SM), as they are used through the E2 interface. The O-RAN Alliance defines four E2SM in its specification: E2 Service Model for Key Performance Measurements (E2SM-KPM) (for the subscription to KPIs and the transmission of KPM reports), E2 Service Model for RAN Control (E2SM-RC) (for the reception of control commands from the Near-RT RIC), E2 Service Model for Network Interfaces (E2SM-NI) (allows the RAN to forward messages obtained at specific network interfaces towards the Near-RT RIC), and E2 Service Model for Cell Configuration and Control (E2SM-CCC) (sometimes also labeled E2SM-CC, enables the Near-RT RIC to reconfigure specific cells or nodes). Furthermore, some Near-RT RIC [42], [44] define custom E2SMs, which can be used if the RAN also implements them. The system encodes E2SMs, along with their messages, into binary using both ASN.1 and Protocol Buffers, while it encodes E2 messages solely in ASN.1. The software leveraged for E2 messaging in both simulators and emulators is further detailed in Section VII.

Finally, the Near-RT RIC provides three more interface terminations: A1, O1, and Y1, (see the violet block in Fig. 3). The A1 interface termination enables the Non-RT RIC to communicate with the Near-RT RIC, allowing the Near-RT RIC to send messages to the Non-RT RIC on its performance or decisions, as well as enabling the Non-RT RIC to change the policies of the Near-RT RIC (see the connection between violet and blue blocks in Fig. 3). The communication with the SMO is done through the O1 interface (see the connection between green and red blocks in Fig. 3), which enables the SMO to deploy or un-deploy xApps. Finally, the latest O-RAN Alliance specifications enable applications, both internal and external to the RAN, to consume Radio Analytics Information (RAI) from the Near-RT RIC through the Y1 interface (see the connection between violet and white blocks in Fig. 3). Specifically, the Y1 is considered a dynamic pathway for enhanced communication and analytics between the Near-RT RIC and both internal and external systems to broaden the scope of O-RAN capabilities towards more adaptive, intelligent, and integrated network operations.

2) Non-RT RIC and rApp: On top of the Near-RT RIC, we find the Non-RT RIC, operating at the largest timescale in the control plane. The Non-RT RIC is also a network element that makes use of KPMs to make control decisions. The Non-RT RIC interacts with the Near-RT RIC, obtaining data of the KPMs stored in the Near-RT RIC (e.g., in the SDL) to perform long-term data analytics over them and take decisions on policies.

This interaction between the Near-RT RIC and the Non-RT RIC is done through the A1 interface.

The Non-RT RIC does not perform the tasks by itself but hosts rApps. Similar to xApps, rApps are microservices that interact with the Non-RT RIC to perform long-term data analysis, ML/AI training, deployment, and un-deployment of xApps, as well as functionalities aimed at operators, including graphical dashboards. Like xApp, rApps are not constrained to specific technologies and are instead general-purpose applications that provide specific functionalities that are useful for the Near-RT RIC. rApps do not have any strict upper bound on response time, and interact with the Non-RT RIC using the R1 interface. As there is no exact boundary between the Non-RT RIC and the SMO, rApps are considered to perform both Non-RT RIC functionalities (e.g., ML training, long-term data analysis) and SMO functionalities (e.g., service management, infrastructural resource allocation).

In terms of architecture, the Non-RT RIC must offer services that enable rApps to perform their tasks. While there is no concrete definition on which services the specification refers to, basic services such as data retrieval and execution of commands from the Near-RT RIC, as well as information gathering from the RAN and the infrastructure using the SMO's interfaces, including the (un)-deployment of xApps. Furthermore, all the services provided by the Non-RT RIC must be offered through the R1 interface, ensuring rApps only connect to the Non-RT RIC through R1 to ensure that rApps developed by various vendors can interoperate seamlessly. Finally, it is key to note from the blue rectangle in Fig.3 that the Non-RT RIC only has two direct standard interfaces: A1 to the Near-RT RIC and R1 to rApps. The interfaces to other elements, such as RAN elements or the infrastructure (O1, O2) are considered a logical part of the SMO. Nonetheless, there is no defined interface through which the SMO provides these interfaces to the Non-RT RIC, so they can be provided to rApps through R1. Instead, it is possible to make the Non-RT RIC assume the responsibilities of the SMO and thus provide these interfaces directly, or alternatively, to consume these interfaces from the SMO using non-standard interfaces.

3) SMO and O-Cloud: The SMO framework is a superset of the Non-RT RIC that does not have a defined boundary with it, as depicted in Fig. 3. The SMO framework is designed to manage and orchestrate the services and resources within

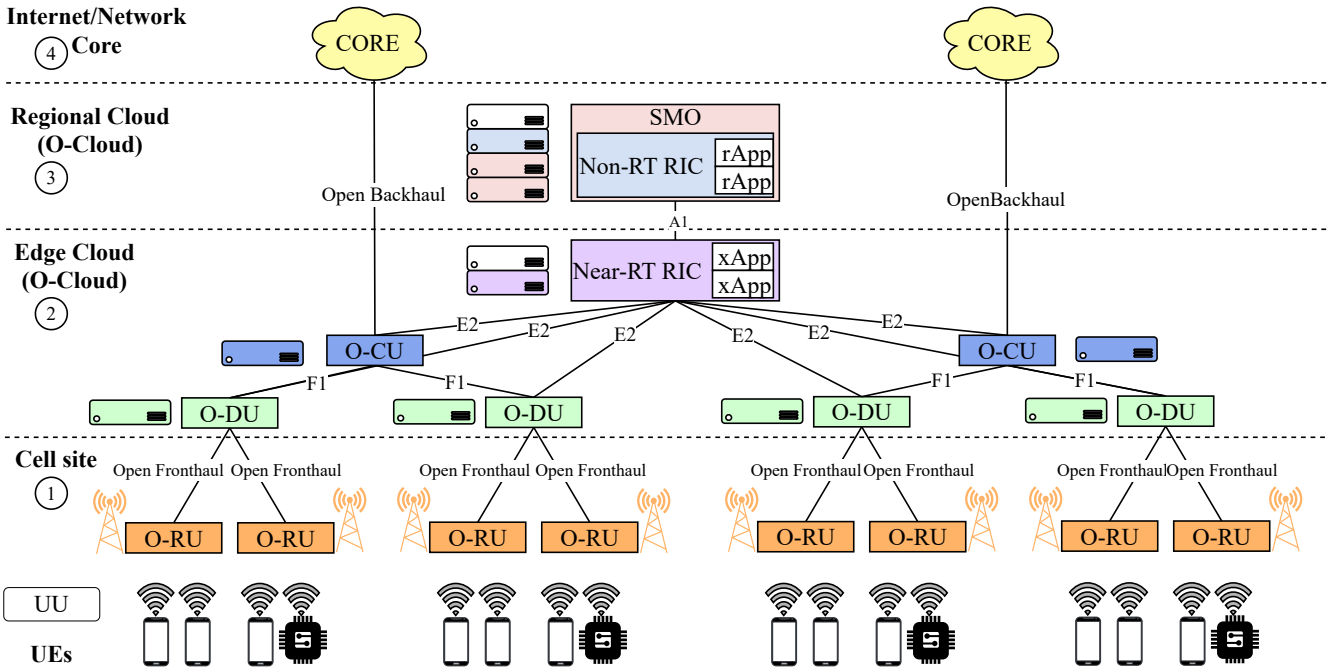


Fig. 4: Example of a complete O-RAN 5G deployment using the 7.2x split that comprises 8 O-RUs, including the main communication interfaces involved.

multivendor O-RAN environments. The SMO's objective is to provide service management and orchestration functionalities, which are pivotal to the RAN. In particular, it enables service provisioning and lifecycle management by acting as a centralized control point for defining, deploying, modifying, and decommissioning services in the O-RAN environment. The SMO also performs resource orchestration as it ensures efficient allocation and utilization of resources. As the SMO communicates with other elements through standard interfaces, it enables multivendor deployments.

An element deeply connected with the SMO is the *O-Cloud*, depicted in grey in Fig. 3. O-Cloud (O-Cloud) is a cloud computing platform encompassing a collection of physical infrastructure nodes that fulfill O-RAN requirements to host relevant O-RAN and SMO functions and software components such as Near-RT RIC and Non-RT RICs, the SMO, O-DU, O-CU-UP, O-CU-CP or the user-plane core. O-Cloud controls resource pools located at one or more sites, alongside software to manage nodes and deployments hosted on them, including both deployment-plane and management services. The O-Cloud architecture consists of three layers: the top layer housing virtualized RAN software functions, a middle layer with Cloud Stack functions, and an Acceleration Abstraction Layer (AAL) that decouples hardware accelerators from their acceleration functions, and a bottom layer constituting the hardware. The O-Cloud architecture allows RAN elements to be hosted in a virtualized environment. Despite the term *cloud*, the O-Cloudinfrastructure can reside at different points in the network, including Edge Clouds, and not necessarily in remote servers.

The SMO can interact with the O-Cloud through the O2 interface. This allows the SMO to operate the functions offered by the O-Cloud, such as infrastructure resource management, deployment management services, or the orchestration and deployment of network functions and services. The services are thus separated between O2 Infrastructure Management Service (O2 IMS), which are related to infrastructure resource management and the storage of inventory resources in a cache, and O2 Deployment Management Service (O2 DMS), which manages O-RAN cloud-native network functions (such as those from the O-DUs and O-CUs) as well as RICs, xApps, rApps, and SMO management functions. To connect to these functions, the SMO framework leverages the O1 interface, where the SMO provides Operational, Administration, and Maintenance, while on the side of the Near-RT RIC, as well as the O-DUs and O-CUs, the O1 interface allows the SMO to collect diverse data from network functions, the Near-RT RIC, and xApps. Nonetheless, as there is no defined boundary across the Non-RT RIC and the SMO framework, the R1 interface also allows rApps to leverage SMO functionalities, and thus, the interplay between R1 and the SMO must be considered. Within the context of the SMO framework, the R1 interface facilitates topology exposure by allowing rApps to access comprehensive network topology information and respond to changes in network topology in real-time. On the other hand, it provides detailed visibility into the network structure and automated inventory management.

C. O-RAN deployment architecture

O-RAN offers a variety of possibilities to deploy each of the entities discussed so far in both the data plane and the control plane within different physical nodes. This freedom in the deployment of O-RAN entities provides two benefits:

first, it allows operators with different physical infrastructures to adopt O-RAN, and second, it allows certain elements to be reduced where they would otherwise be redundant. This feature is often known as *gNB disaggregation*, as not only the network functions of each gNB are logically disaggregated into different elements, but they can also be physically disaggregated into multiple devices. The physical disaggregation is especially interesting when seen from an O-Cloud perspective, as the elements become cloud-native applications that can be deployed, orchestrated, and managed using state-of-the-art platforms.

To better illustrate a disaggregated deployment, Fig. 4 depicts an example of a 5G RAN that comprises 8 O-RUs that serves 16 UEs with an O-RAN architecture based on the 7.2x split. In this example, there are four different physical locations where elements can be deployed: the **cell sites** (Layer 1 in Fig. 4), which refer to the physical locations where the antennas are placed; the **edge cloud** (Layer 2 in Fig. 4), a set of dedicated nodes very close to cell sites; the **regional cloud** (Layer 3 in Fig. 4), which includes nodes in data centers in the same cloud region as the cell site; and the **network core** (Layer 4 in Fig. 4), which allows for connections to remote locations through the Internet. In this example, the latency between the cell site and the *edge cloud* is bounded below 5 ms, while the latency between the *edge cloud* and *regional cloud* is approximately 50 ms, and the latency to the *network core* depends on the machine that is reached. Beginning from the bottom, we find the UEs, usually located in the *cell-site* area, which uses mmWave technology to connect to the antennas. Integrated with the hardware of the *cell site* (Layer 1 in Fig. 4), we find the O-RU, where each antenna is coupled with its own O-RU. These O-RUs then connect to the *edge cloud* (Layer 2 in Fig. 4), where other O-RAN elements are deployed. Unlike O-RUs, each antenna does not have a whole dedicated O-DU; instead, there are 4 *edge nodes* where O-DUs are deployed, where each O-DU serves two O-RUs. Nonetheless, these *edge nodes* implement only the O-DUs, as the O-CUs are deployed to two separate nodes that communicate with the O-DUs through the F1 interface. In this example, the O-CUs are *joint*, i.e., implement both O-CU-UP and O-CU-CP functionalities. To finish the user plane, the O-CUs connect to the core, residing in the network core, through the Backhaul interface.

Focusing on the control side, the Near-RT RIC is deployed to a different device in the *edge cloud* (Layer 2 in Fig. 4), to which O-CUs and O-DUs connect through their E2 interfaces. As the latency of the *edge cloud* is relatively small (5 ms), it is possible to perform the control loop within the 10-1000 ms timescale of near-RT. Along with the Near-RT RIC, xApps are deployed to a co-located device, where each xApp consumes the services offered by the Near-RT RIC. On top of the Near-RT RIC, we find the SMO and Non-RT RIC, deployed to the *regional cloud* (Layer 3 in Fig. 4), and communicating through the A1 interface. The SMO and Non-RT RIC's services are deployed throughout 3 different servers in the *regional cloud* (Layer 3 in Fig. 4), while a 4th server hosts the rApps, which uses the R1 interface to connect with the SMO. Finally, both the *regional and edge clouds* are part of the O-Cloud platform, and thus, the SMO can manage them. It is important to note that the figure only represents the main connections, and other secondary interfaces (e.g., O2, O1, Y1) are not depicted.

D. DevOps tools for the RAN

DevOps is a collaborative approach that combines software development and IT operations to speed up the system development lifecycle. It is becoming increasingly popular in 5G networks due to the challenges of managing and controlling the code lifecycle from development to operations phase in distributed scenarios. This approach enhances the frequency of feature delivery, fixes, and updates while aligning with business objectives. These principles include automation, monitoring, and collaboration throughout all development stages, coinciding with the O-RAN Alliance specifications [2].

The DevOps pipeline for a RAN project starts with the *plan* phase. During this phase, the project requirements are defined by studying the scope and objectives of the 3GPP standard, O-RAN Alliance specifications, and SCF ones. This ensures that scalability, security, and maintenance considerations are integrated from the beginning. Tools like Jira [46] and Confluence [47] aid in progress management. Following planning, the *code* phase involves developers writing code in a shared repository, utilizing version control systems like Git [48] for collaboration and Continuous Integration (CI) tools such as Jenkins [49] to facilitate regular testing and merging. The coordination of this phase is critical for most 5G RAN projects, considering the numerous participants in each open-source project [50]. The *build* phase transforms code into deployable artifacts through automation tools like CMake [51], Gradle [52], and Maven [53], adhering to the principle of "build once, deploy anywhere." Build tools are crucial as they implement dependency management systems, which help to avoid code vulnerabilities through automated scanning, prevent version pinning that restricts updates, verify signatures and hashes, generate lock files with compiled dependencies, and remove unused dependencies during code refactoring. The *test* phase is integrated into the CI/Continuous Deployment (CD) (CI/CD) pipeline to catch bugs early using automated tools like GoogleTest [54] and JUnit [55], while manual testing remains necessary for complex scenarios. The goal is to ensure quality before production deployment to maintain up-to-date dependencies and early identify bugs rising from the deployment of the RAN Project [56]. The *release* phase involves scheduling the build-tested code for deployment in a real testbed. In this phase, automated continuous delivery pipelines deploy the software into live environments using tools like GitLab Continuous Integration (CI)/CD [57], minimizing human intervention and reducing error risks. The *deploy* and *operation phases* generally use Infrastructure as Code tools to provision infrastructure, deploy applications, configure the application and platform, and orchestrate the environment. The *deploy* phase may occur incrementally or all at once, with continuous deployment automatically pushing code changes to production after passing tests, facilitated by orchestration tools, such as Kubernetes [58]. The release and deployment phases of 5G RAN projects are crucial due to the distribution

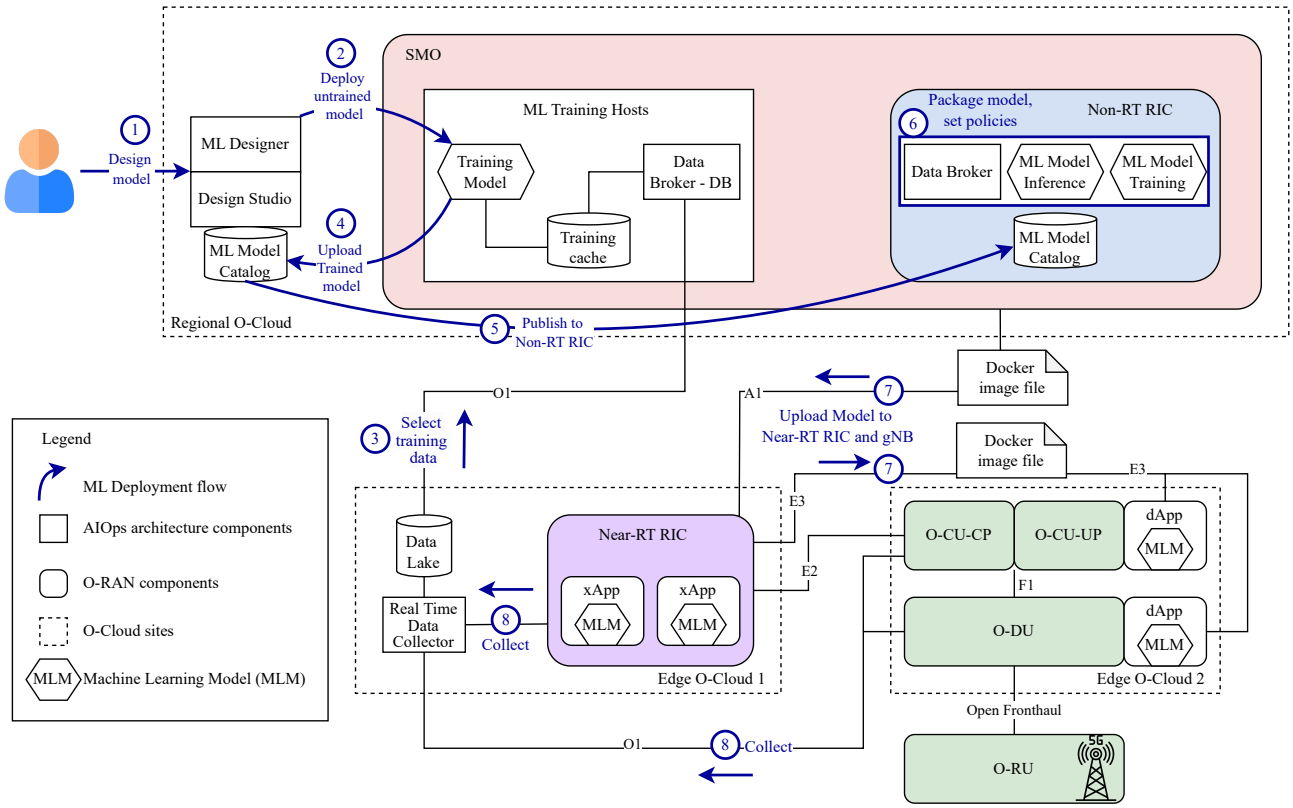


Fig. 5: AIOps architecture of O-RAN Alliance for the distribution of ML/AI workflows in the RAN

of artifacts across heterogeneous sites, where infrastructure differences can lead to errors regarding the incompatibility of the isolation technology of the artifacts, different infrastructure configurations, or even different drivers [59]–[61]. Thus, the execution in simulated, emulated, or real environments can require significant configuration adjustments. In the *operate* phase, real-time monitoring tools such as Prometheus and Grafana help ensure application performance and uptime, allowing teams to address issues proactively [62], [63]. Finally, in the *monitoring* phase, user feedback and monitoring data inform future improvements, creating a continuous loop of enhancement. This phase involves controlling changes in the deployed DevOps pipelines. These pipelines should be monitored based on data collected from the 5G RAN deployments and their performances [64], [65] to provide feedback on the overall system. Overall, the DevOps lifecycle emphasizes collaboration, automation, and ongoing feedback, enabling organizations to enhance software quality and responsiveness to change. Moreover, the challenges that are still open to fully integrating DevOps in the RAN are discussed in Section VIII.

E. AI and ML for the O-RAN architecture

The recently founded AI-RAN Alliance explores various applications and the coexistence of RAN with AI, with the objective of proposing new architectures and specifications to enhance the performance of RAN and unlock new capabilities and business opportunities [66]. AI optimization for the RAN primarily intervenes in the automation of control loops. The integration of ML models inside xApps and rApps requires an efficient Machine Learning Operations (MLOps) architecture to sustain the training, testing, and validation of the models [67], and to distribute the models within the RAN deployment architecture [68].

At the moment of writing, the O-RAN Alliance proposes a first example of lifecycle management of ML applications, presenting key phases involved in the design and deployment within the O-RAN architecture. Figure 5 illustrates the typical steps in applying AI/ML-based use cases in O-RAN, considering both supervised and unsupervised learning ML models. The process begins with the ML modeler using a designer environment along with various ML toolkits to create the initial model. This model is then sent to training hosts for training, with appropriate data sets collected from the Near-RT RIC, O-CU, and O-DU, stored in a data lake, and passed to the training hosts. After training, the model or sub-models are uploaded to the ML designer catalog, where the final model is composed. The trained ML model is then published to the Non-RT RIC along with associated licenses and metadata. The Non-RT RIC can create a containerized ML application with the model artifacts and deploy it to the Near-RT RIC, O-DU, and O-CU using the O1 interface, with policies set through the A1 interface. Additionally, Performance Management (PM) data can be sent back to the training hosts from the Near-RT RIC, O-DU, and O-CU for potential retraining. Notably, the Near-RT RIC can also update the ML model parameters

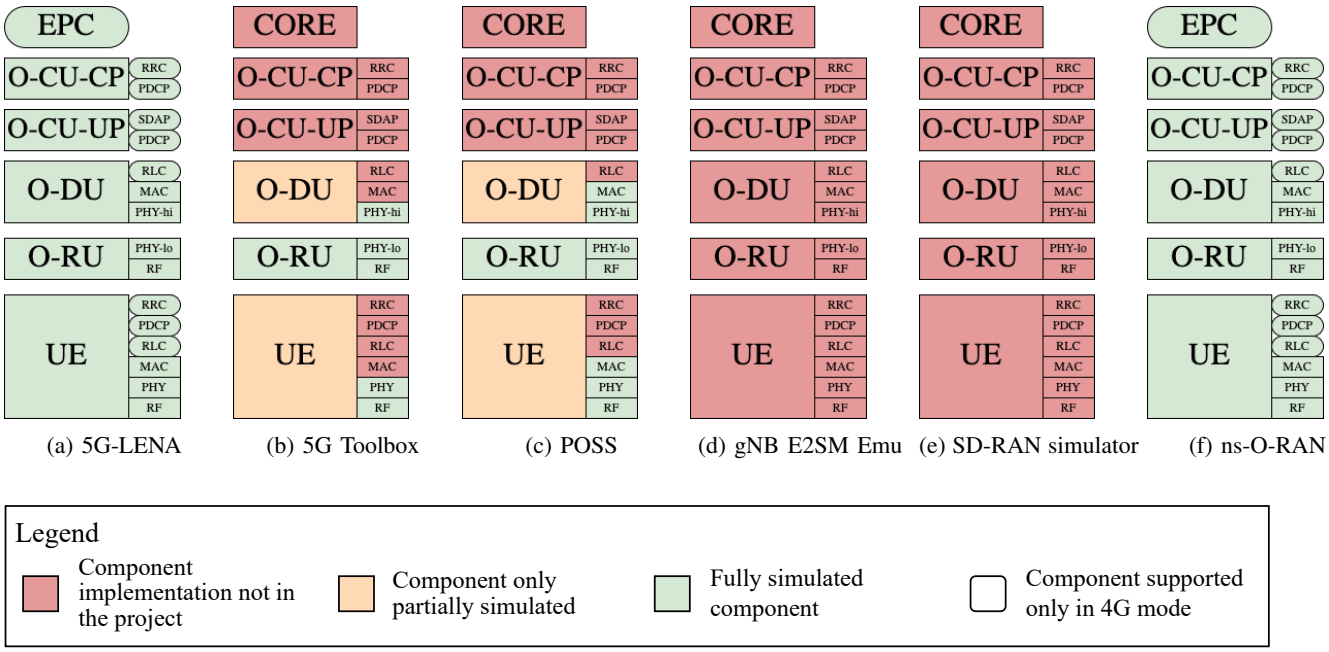


Fig. 6: Visual architecture of each simulator, showing the simulated data plane components. Green components and network functions are simulated, orange ones are partially simulated, and red ones are not. Rounded components are only supported in 4G mode. The legend below explains the color coding.

at runtime without the need for extensive retraining, with both training hosts and ML designers potentially being part of the Non-RT RIC.

IV. O-RAN Network Simulators

O-RAN network simulators are tools that can be of great importance for research and industry and improve the adoption of O-RAN. On the one hand, simulators provide an environment to test the performance of O-RAN elements in a variety of scenarios without the need for specialized hardware, especially in cases in which it may be unfeasible or impractical to recreate the scenario using real hardware (e.g., large scale scenarios). On the other hand, simulators ensure the replicability and comparability of the obtained results and the tested algorithms, methods, or elements. For example, if a certain handover algorithm is tested with a simulator in a given scenario, it is possible to replicate the scenario by configuring the simulator in the same manner (i.e., using the same scenario specification), and thus, it is possible to compare it with other algorithms in the same scenario, which may be more complicated with real elements. Moreover, some simulators provide APIs compliant with O-RAN standards, mainly in the form of E2 interfaces, and are thus interoperable with real O-RAN software. Due to the higher maturity and technology readiness of Near-RT RICs and xApps [69], this section focuses on them rather than on Non-RT RICs and rApps. The interoperability with O-RAN software gives the simulators an additional role, as validators and evaluators of O-RAN software through integration tests. Following the prior example, instead of testing two handover strategies implemented in the simulator, it would be possible to test two xApps implementing handover strategies directly. In the remainder of this section, we analyze the current state of the art in terms of O-RAN network simulators.

A. 5G-LENA

The first simulator we review is 5G-LENA [70], a simulator of 5G New Radio (NR) networks developed by the Mobile Networks group of the Centre Tecnològic de Telecomunicacions de Catalunya (CTTC). 5G-LENA is based on the ns-3 simulator, a state-of-the-art network simulator based on discrete event simulation and programmed in C++ [70]. Concretely, 5G-LENA makes use of the existing tools of ns-3, including its distributed event simulation core, libraries, workflow, interface for simulation configuration, or methods for trace collection, and includes its functionalities on top of the ns-3 simulator. It is, thus, a ns-3 *module*, that can be added to an existing instance of ns-3 to provide 5G simulation capabilities, as long as the other modules of said ns-3 instance are not incompatible [70]. 5G-LENA was built as an evolution of LENA, a similar ns-3 module for the simulation of LTE networks, and its code started as a fork of the ns-3-mmWave module [71], which handles the simulation of 5G and LTE networks. From a high-level perspective, 5G-LENA can be seen as an alternative to the ns-3-mmWave module with a greater focus on 5G networks. 5G-LENA provides simulation at different layers: PHY, MAC, RLC, and PDCP, as well as the core network [70], as shown in Fig. 6a. The PHY and MAC layers have been entirely developed for the use of NR networks, while the RLC, PDCP, and core network features are directly reused from LENA's LTE implementation. This is especially noteworthy in the case of the core, as only the LTE core EPC is supported in 5G-LENA.

In terms of O-RAN network simulation, however, 5G-LENA has a key limitation, as support for the O-RAN interfaces for control (e.g. E2, O1) is still a work in progress [72]. While 5G-LENA provides extensive and detailed simulation models for the 5G data plane, including the simulation of gNBs and UEs from the PHY layer to the PDCP layer, as well as simulation for the core network, it does not provide for such extensive control plane simulation. Hence, the O-RAN control plane cannot be simulated through 5G-LENA, as it does not provide an interface to communicate *online* with a Near-RT RIC, nor a simulated, *offline* Near-RT RIC that can be programmed or configured within the simulations. This limitation may not be of importance in use cases that only need the data plane to be simulated (e.g., traffic trace obtention), but can hinder its use for the validation or evaluation of Near-RT RICs or xApps. In summary, 5G-LENA provides extensive simulation of the lower layers of the data plane for 5G networks, which supports the EPC core network but lacks support for the management of the 5G network through the O-RAN control plane.

B. 5G Toolbox

The 5G Toolbox [73] is a commercial simulator for 5G networks developed by Mathworks and integrated with MATLAB software, simulating the elements shown in Fig. 6b. The 5G Toolbox is aimed at the simulation and validation of 5G systems, including features such as waveform generation, link-level simulation, MIMO, and beamforming. 5G Toolbox supports the simulation of arbitrary scenarios where multiple UEs share a network, as well as the direct application of AI technique to the wireless 5G operation. However, the 5G Toolbox limits its simulation, including the training and usage of AI models, to the PHY layer. Due to its narrow scope, the 5G Toolbox can only be considered as an alternative if the simulation does not include elements beyond this layer. Moreover, this PHY layer simulation does not account for the usage of a real or simulated Near-RT RIC, and instead, the parameters and techniques used, including those based on AI, must be applied directly on the network, not through xApps. The 5G Toolbox enables the development of algorithms that can generate data for fronthaul and other O-RAN interfaces.

C. POSS

The Python Open-Source Simulator (POSS) [74] is a 5G network simulator developed by the University of Athens. The objective of POSS is to simulate key KPIs of a RAN, from the RF to the MAC layer, and use their simulated KPMs to create a dataset. This dataset can then be used to train AI-based systems, with a special focus in Deep Reinforcement Learning (DRL) models. POSS uses mathematical models to approximate the signal quality parameters like Received Signal Strength Indicator (RSSI), Reference Signal Received Power (RSRP), Reference Signal Received Quality (RSRQ), Channel Quality Indicator (CQI), and throughput that different UEs experience in a given simulation, and can also simulate the movement of users and their association to different gNBs. Furthermore, POSS enables some of its configuration parameters, such as the power budget or 5G numerology, to be changed by external xApps, thus allowing the integration of real xApps, and especially DRL models, with POSS. Its main limitation in terms of simulation is that POSS does not support simulated traffic in its network: traffic-related KPMs, such as throughput, are evaluated based on their *potential* maximum, as UEs are not sending nor receiving any traffic. POSS thus simulates the RF, PHY, and MAC layers, not functionally, but in terms of KPIs/KPMs, but the rest of the layers, as shown in Fig. 6c, are not supported.

In terms of the control plane, POSS has a different implementation: while it can be integrated with Near-RT RICs and other xApps, this integration is not done using the E2 interface. Instead, POSS works as an xApp, that communicates with other xApps using the services provided by the Near-RT RIC. This implementation further complicates the usage of POSS, as it needs a Near-RT RIC that supports xApps to send KPM messages and receive control messages without the intervention of the RAN, possibly requiring a custom Near-RT RIC for the task. Moreover, the usage of POSS can be problematic when xApps are to be migrated from the simulation to a real testbed, as they will communicate through different means: xApps tested with POSS will expect data from other xApps, while those developed for real testbeds will expect data from the RAN. Finally, the main flaw of POSS is that the authors do not provide an implementation, as the repositories linked in the paper no longer exist. Although the evaluation of POSS includes its deployment as a Docker container [74], neither the Docker image leveraged, its Dockerfile and the necessary files to build it, nor any other analogous container definition that could allow its replication, are public.

D. gNB E2SM Emu

gNB E2SM Emu [75] is an O-RAN simulator programmed in C, intended for the testing of Near-RT RICs and xApps against an E2 interface. gNB E2SM Emu simulates a single gNB servicing 4 UEs, providing an E2 interface for the Near-RT RIC to connect to. The simulator assumes that messages are encoded using Protocol Buffers, providing support for indication requests and control messages. However, simulation requests can only ask for the ID of the gNB or the UE list. In the case of the latter, gNB E2SM Emu responds with random KPMs for each UE, as well as UE-specific properties. Both UE properties and the gNB's ID can be changed through control messages. Overall, gNB E2SM Emu provides a simplistic simulation of an E2 interface and allows for the validation of Near-RT RICs and xApps. However, its lack of a proper service model, relying on a custom Protocol Buffers definition, as well as the random nature of the KPMs obtained and its inability to simulate the RAN's data plane, as depicted in Fig. 6d, limit its usefulness for complete RAN simulation.

E. SD-RAN simulator

The SD-RAN simulator [76] is an O-RAN-compliant simulator intended to serve as a testing environment for the development of xApps. SD-RAN simulator enables the simulation of environments with an arbitrary amount of gNBs and UEs, including the movement of the users holding the

UEs, the cells simulated and their coverage, and the number of cells served by each gNB. These simulation scenarios can be created manually or generated automatically given certain parameters (e.g., number of gNBs, number of UEs) through the accompanying *Honeycomb Topology Generator* tool. SD-RAN simulator can be integrated with Near-RT RICs through an E2 interface shared by all gNBs in the simulation. Nonetheless, the SD-RAN simulator allows each gNB to be controlled by a different RIC, if necessary. Specifically, the SD-RAN simulator is built for its use with the μONOS RIC, which is also part of the SD-RAN project, although multiple instances of the μONOS RIC can be used to control different parts of the RAN if necessary. The simulator also allows the user to configure the service models enabled at each gNB out of those supported, which include various versions of KPM and RAN Control (RC) as well as non-standard service models used by the μONOS RIC. As the SD-RAN simulator implements the RC service model, it can be controlled by xApps sending the appropriate commands, as well as through the RAN Simulation API and UE Simulation API, which are provided as part of the μONOS Command Line Interface and can perform direct modifications in the gNBs and UEs simulated (e.g., adding or removing gNBs from the simulation). It is noteworthy that these APIs refer to simulation-specific control, and hence, should not be considered part of O-RAN.

The main limitation of the SD-RAN simulator, however, is its lack of support for traffic: although it does simulate the control plane through the E2 interface, the UEs in the simulation are only simulated in terms of movement and do not implement the user plane, neither in terms of the core network nor any traffic across UEs, a limitation clearly shown in Fig. 6e.This greatly limits any use cases requiring traffic QoS metrics, although those focusing exclusively on handover can be used. It is also noteworthy that control commands cannot be sent using the standard RC service model, and instead, the non-standard Mobile HandOver (MHO) service model must be used for this task. Overall, the SD-RAN simulator can be a good option for use cases that are not focused on obtaining KPI due to its ease of use and API provision, also its lack of support for a simulated data plane makes it unsuitable for obtaining and testing traffic-related metrics. On the other hand, if the SD-RAN simulator is seen as a tool to validate xApps and Near-RT RICs in development, it is a more complete alternative to gNB E2SM Emu.

F. ns-O-RAN

ns-O-RAN [77], proposed by Lacava *et al.*, is an O-RAN simulator that is also based on ns-3. ns-O-RAN provides a modified version of the ns-3-mmWave module[1], which handles the simulation of 5G and LTE networks using mmWave frequencies, including some of their key aspects, e.g., PHY and MAC layers, MAC scheduling, carrier aggregation, handover, or core network elements, as depicted in Fig. 6f. However, neither the original nor the modified mmWave modules provide support for the O-RAN E2 interface. To enable messaging through the E2 interface, ns-O-RAN adds the *O-RAN E2 interface* module[2] to ns-3, which provides each simulated gNB with its own E2 termination and interface, as well as an implementation of the KPM and RC service models in ASN.1 format. ns-O-RAN runs in a single Docker container or computing node, but the simulation may contain multiple gNBs, and thus, the E2 terminations provided by the O-RAN interface modules are implemented to be multiplexed over a single E2 termination, that the container can then expose. This simulated E2 interface, modified to enable its multiplexing to the gNBs, is implemented through the use of a modified E2SIM library[3].

As ns-O-RAN is based on ns-3, it is possible to simulate not only scenarios with an arbitrary number of gNBs and UEs but also to specify a wide variety of parameters for them (e.g., location, mobility model, bandwidth, number of antennas, generated traffic). Furthermore, ns-O-RAN provides a multiplexed but compliant E2 termination, enabling its use with Near-RT RICs and xApps that leverage the KPM or RC service models, rather than only simulated Near-RT RICs or xApps, hence contributing to bridging the gap between simulated and real testbeds. However, ns-O-RAN has certain limitations, especially in terms of supported KPIs and control parameters. Currently, ns-O-RAN only supports the use of control messages to perform handover operations and simulates a subset of the RAN's KPIs. In conclusion, ns-O-RAN is, thus, a simulator for O-RAN networks that can be used for purposes such as testing and experimentation on real Near-RT RIC and xApp software as long as the simulated scenarios and use cases fall within its limitations.

G. Lessons learned

In general, evaluated simulators provide tools for simulating the E2 interface in each gNB, the data plane of 5G networks, or even both, aimed at different use cases. As such, they should be compared by their main features, which can be advantageous or disadvantageous for different use cases. In this context, 5G-LENA is a precise simulator of the O-RAN data plane without control plane support, whereas 5G Toolbox is aimed at very advanced simulations of the RF and PHY layers exclusively. POSS is a simple but non-orthodox simulator for the data plane with control plane support. gNB E2SM Emu features easy integration testing of E2 interfaces without adhering to a specific control plane, a similar functionality to SD-RAN simulator, which serves this purpose for μONOS-based control plane. Finally, ns-O-RAN provides data plane simulation with limited support for integrating with real control plane implementations.

Based on these features, we advise developers to use different simulators for the following use cases. For the validation of Near-RT RIC and xApps by connecting to an E2 interface, gNB E2SM Emu is a simple but effective simulator, while the SD-RAN simulator provides a more complex but also more comprehensive model. If the use case is aimed exclusively at the data plane, 5G-LENA supplies the most extensive simulation model. Nonetheless, 5G Toolbox provides a more

[1]https://github.com/wineslab/ns-oran-ns3-mmwave

[2]https://github.com/oran-sc/sim-ns3-oran-e2

[3]https://github.com/wineslab/ns-oran-e2-sim

TABLE II: Comparative table of the simulators

	Feature	5G-LENA [70]	5G Toolbox [73]	POSS [74]	gNB E2SM Emu [75]	SD-RAN simulator [76]	ns-O-RAN [77]
General	**Base simulator**	ns-3	MATLAB	None	None	None	ns-3
	Multiple scenario support	Yes	Yes	Yes	No	Yes	Yes
User Plane	**5G simulated layers**	PHY, MAC	PHY	RF, PHY, MAC	None	None	PHY, MAC
	4G simulated layers	RLC, PDCP, SDAP, RRC	None	None	None	None	RLC, PDCP, SDAP, RRC
	Supported cores	EPC	None	None	None	None	EPC
Control Plane	**E2 interface support**	No	No	No	Yes	Yes	Yes
	Supported Service Models	None	None	None	None	KPM, RC, NI, μONOS' non-standard service models	KPM, RC
	Data encoding	None	None	None	Protocol Buffers	ASN.1, Protocol Buffers	ASN.1
	KPIs sent through E2	None	None	None	Random KPMs	Static KPMs	UE-specific KPIs (PRBs allocated, MCS tx, SINR per tx, MAC PDUs, MAC volume, QAM/QPSK, buffer occupancy, downlink throughput, PDCP PDUs transmitted and split, SINR of serving cell, SINR of best neighbour cell, cell ID)
	Parameters controllable through E2	None	None	None	gNB ID, UE properties	Cell offset, hysteresis	Handover
DevOps	**Plan: Code planning, collaboration, version**	None, Google Group, GitLab	None, MATLAB Answers, None	None, None, None	None, None, GitHub	None, GitHub Discussions, GitHub	None, Gerrit and GitHub Discussions, GitHub
	Code: Main language	C++	MATLAB	Python	C	Go	C++
	Build: Dependencies	10 dependencies	1 dependency	Unknown	1 dependency	24 dependencies	11 dependencies
	Build: compiler	CMake (g++)	None	CPython	CMake (g++)	Go	CMake (g++)
	Test: code testing	Unit test available	Not available	Not available	Not available	Unit test available	Unit test available
	Release: CI/CD tools	GitLab CI	None	None	None	GitHub Actions	None
	Deploy: Reference OS	Ubuntu (vN.A)	Windows, Linux, macOS	Windows, Linux, macOS	Ubuntu (v20.04)	Ubuntu (v22.04)	Ubuntu (v18.04)
	Deploy: Containerization	No	No	No	Yes, Dockerfile	Yes, Docker image	Yes, Dockerfile
	Deploy: types	Bare Metal	Bare Metal	Bare Metal	Bare Metal, Docker Compose	Bare Metal, Kubernetes, Helm	Bare Metal, Docker
	Operate: Configurations	Using CLI parameters, custom C++ scenarios	Using MATLAB scenarios	Unknown	None	Using μONOS config CLI	Using CLI parameters, custom C++ scenarios
	Monitor: tools	ns-3 runtime logs, ns-3 traces	MATLAB API	Unknown	Runtime logs	μONOS monitoring, runtime logs	ns-3 runtime logs, E2Sim runtime logs enabled at build time, ns-3 traces

precise model if only RF and PHY are considered. For use cases such as xApp validation, or general-purpose simulation, we recommend ns-O-RAN, as it is the most complete simulator currently available: it provides not only an E2 interface but also a complete simulation of the data plane, enabling the previous use cases as well as the experimentation and evaluation of real Near-RT RICs and xApps in a simulated RAN. However, it is essential to note that the simulation of some features, such as the 5GC, the management of 5G control parameters (e.g., slicing) from xApps, or the disaggregation of Centralized Unit (CU), Distributed Unit (DU), and RU elements, are not yet supported by the current state-of-the-art simulators. Most of these simulators can be executed through virtualization, easing their management, although completely automated orchestration and deployment for O-RAN simulators are still a work in progress. Moreover, the focus on xApps and the Near-RT RIC stems from the comparatively lower maturity and technology readiness level of the Non-RT RIC [69]. A detailed comparison of all the characteristics supported by all the analyzed simulators is available in Table II. It is noteworthy that the characteristics in Table II are analyzed based on their official releases. While the authors acknowledge developers can modify these simulators to include support for some of these features (e.g. containerization), as an effort must be made by developers to support them, they are considered unavailable or unsupported.

V. O-RAN Emulators and Testbeds

This section overviews existing emulators and real-world tools implementing the 5G RAN. The decision to combine both emulation and real testbed environments stems from the availability of tools that offer dual functionalities in certain cases. The section is organized into subsections, each dedicated to an in-depth analysis of the following tools: O-RAN Software Community (O-RAN SC) [50], Software Radio System RAN (srsRAN) [78], OAI [79], Simu5G [80], and FikoRE [81]. Each project is examined in a structured manner, beginning with a presentation of its key control and data

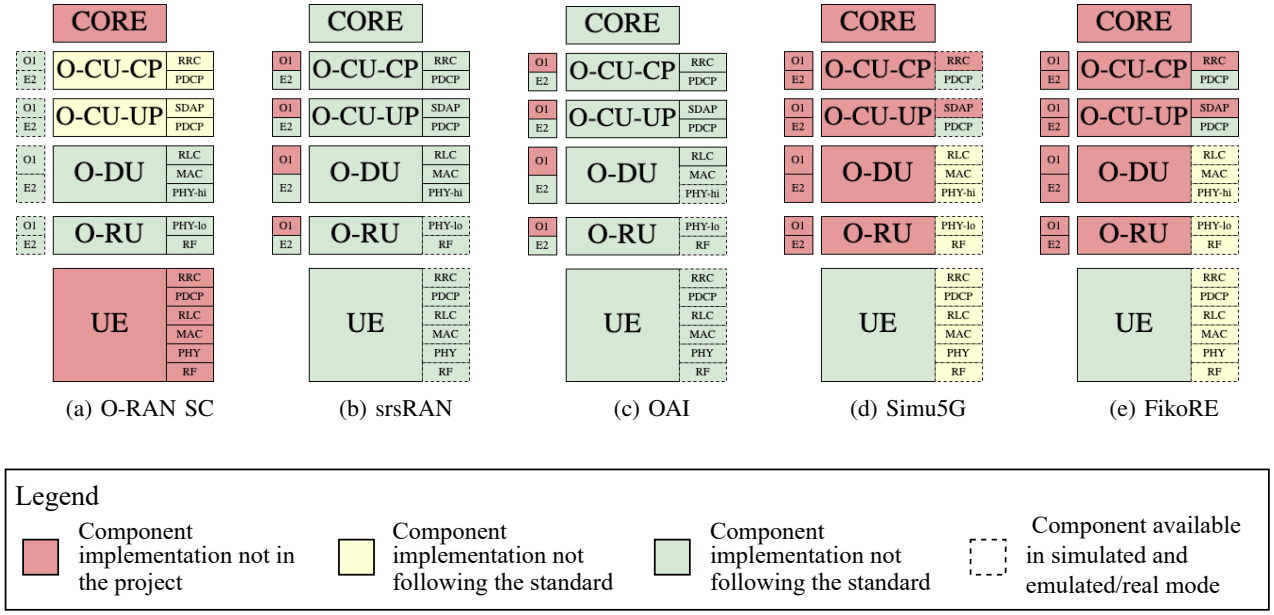


Fig. 7: Visual architecture of each emulator and real testbed, showing the data plane and control plane components. Green components and network functions are implemented and follow the standard, yellow exist but do not follow the standard, and red ones are not part of the project. Dashed components exist both in simulated mode and in emulated or real mode. The legend below explains the color coding.

plane features, followed by an analysis of its architecture, and concluding with a discussion of the adopted DevOps tools available to users.

A. O-RAN Software Community (O-RAN SC)

O-RAN SC is an open-source implementation of the RAN supported by the O-RAN Alliance and Linux Foundation that fosters the development of an open RAN solution available to everyone. It follows both the 3GPP standard and the O-RAN Alliance specifications and can execute either as an emulator or in a real testbed. The O-RAN SC project implements only parts of the gNBs; in particular, only the implementations of the O-DU, and O-CU are available, as depicted in Fig. 7a. The O-CU project was under development. However, it is currently on hold and no longer in progress.

The O-DU is divided into two projects named O-DU Low and O-DU High, where the first comprises layer 1 of communication, and the second comprises layer 2. The O-DU low project focuses on the baseband PHY Reference Design and leverages processors with Intel Architecture to execute. Layer 1 has three interfaces to communicate with other network functions: the interface between layer 1 and Open Fronthaul adopts the WG4 specification for the Control User Synchronization planes communication [82]; the interface between O-DU Low and O-DU High adopts the FAPI interface according to the O-RAN Alliance WG8 specification [83]; interface between O-DU Low and accelerator employs DPDK BBDev acceleration [4] following the WG6 of O-RAN Alliance [84].

To grant compatibility with different O-RU it leverages Intel FlexRAN [5] that implements the various channel models at the PHY layer adhering to the 3GPP standard.

The O-DU High implements the functional blocks of the protocol stack in Stand Alone (SA) mode, primarily including MAC, Scheduler, and RLC layers. The O-DU High architecture is delineated at a thread level, where each thread serves a distinct function. Thread 1 operates the main O-DU thread, while Thread 2 encompasses the DU APP, which includes the Config Handler, O-DU Manager, UE Manager, and ASN.1 Codecs. Thread 3 handles the 5G RLC downlink and MAC, including the 5G Scheduler and Lower MAC, whereas Thread 4 is dedicated to the 5G RLC uplink. Thread 5 manages the SCTP Handler, Thread 6 oversees the Lower MAC Handler, and Thread 7 is responsible for the General Packet Radio Service (GPRS) Tunnelling Protocol (GTP) Handler. Thread 8 handles the O1 interface.

The O-RAN SC project supports the 7.2x split and can connect to the rest of the RAN with the E2 and O1 interfaces. It is not compatible with 4G components, and it has not been tested yet with core networks. The O-DU low strongly relies on Intel processors, while the O-DU High implementation is deployment agnostic and does not require special changes to be used in virtualized or container-based deployment options. Given the involvement of numerous contributors, the O-RAN SC project places significant emphasis on DevOps tools. Specifically, they integrate code deployment processes using Jenkins, Gerrit Repos, and Confluence. However, these tools are primarily focused on managing the internal code lifecycle before it is released to the public repositories.

[4]https://dpdk-power-docs.readthedocs.io/en/latest/bbdevs/

[5]https://github.com/intel/FlexRAN.git

B. Software Radio System RAN (srsRAN)

srsRAN is an open-source implementation of 5G UE/O-DU/O-CU for emulation and real testbeds [85]. It complies with 3GPP standard release 17 and O-RAN Alliance specifications, supporting backward compatibility with 4G, and has tested with core networks like Open5GS and Free5GC. The project includes all physical channels and enables RFs simulation with Multiple Input Multiple Output (MIMO) capabilities. Additional features include network slicing, hardware acceleration, and Non-Terrestrial Network (NTN) GEO support.

srsRAN supports both split-8 and split-7.2x fronthaul, with split-8 enabled via USRP Hardware Driver (UHD) for Universal Software Radio Peripheral (USRP) devices and split-7.2x via DPDK. Its architecture (shown in Fig. 7b) includes the following components O-CU-CP, O-CU-UP, O-DU-high, O-DU-low, and Open Fronthaul, all implemented in software and customizable. Users can integrate third-party RICs, O-RUs, and gNB components.

The O-CU-CP is responsible for handling control plane messaging, specifically the PDCP control plane aspect of the protocol. It comprises five main components: the O-CU-CP Processor, O-DU Processor, UE Manager, Measurement Manager, and Mobility Manager. These components manage connections to multiple O-CU-UPs, handle O-DUs, manage UEs, oversee cell measurements, and manage UE mobility, respectively.

The O-CU-UP manages user plane messaging, handling PDCP and SDAP protocols. It includes four components: the UE Manager (managing connected UEs), GTP-U (handling User Plane traffic to/from the UPF in the 5GC via the N3 interface), SDAP (mapping QoS requirements), and PDCP (processing User Plane data before it enters the O-DU-high). O-CU-UP interfaces with the 5GC via N3, O-CU-CP via E1, O-DU-high via F1-u, and can connect to the Near-RT RIC through E2.

The O-DU-high manages uplink and downlink traffic, focusing on MAC and RLC processing. It comprises three components: the O-DU Manager (overseeing the O-DU, connected UEs, and RAN resources), the RLC layer (providing transport services to the O-CU), and the MAC layer (encoding/decoding MAC PDUs and scheduling grants). O-DU-high interfaces with the O-CU-CP and O-CU-UP via F1-c and F1-u, O-DU-low via FAPI, and connects to the Near-RT RIC through E2.

The O-DU-low manages uplink and downlink traffic through Upper PHY processing. It consists solely of the Upper PHY and has two interfaces: one with O-DU-high via FAPI and another with the O-RU via Open FrontHaul.

srsRAN [85] is written in C++17 and can be compiled with gcc (v9.4.0 or later) or Clang (v10.0.0 or later). The platform requires several build tools and libraries for proper functionality, with CMake for managing the build process. Components can be configured through the provided YAML files. srsRAN can also run as Docker containers in Kubernetes, with Dockerfiles available to build component images for Helm deployments. The project is well-documented, offering recommendations for source code formatting, tutorials for network configuration deployment, troubleshooting guidelines, and an issue-reporting page. Code development is supported by GitLab CI for continuous integration.

C. Open Air Interface (OAI)

OAI is an open-source implementation of 5G that complies with 3GPP standards and O-RAN Alliance specifications [86]. OAI provides a comprehensive platform for both 4G and 5G NR technologies, suitable for emulation and real testbed. Figure 7c shows the OAI software architecture.

The gNB at the PHY layer supports various numerologies, channel models, advanced MIMO configurations, and beam management. The MAC layer provides flexible structures and dynamic resource scheduling, while the RLC layer handles segmentation and error correction. The PDCP layer includes header compression and ciphering, and the RRC layer manages connection procedures and mobility. The NR UE features a complete protocol stack supporting mobility, measurement reporting, and beam management for 5G NR deployments.

The RFs simulator replaces the radio board with software (TCP/IP) communication, enabling functional tests without an RFs board. The OAI gNB and UE interact as if an RFs interface were present. The Level 2 simulator connects the UE with the eNB and gNB via nFAPI, allowing multiple simulated UEs to connect to the MAC stub.

In OAI, a O-CU O-DU split version of the 5G gNB deployment is available and has been validated in 5G SA mode with the OAI RFs simulator. This split mode facilitates control plane exchanges between the O-CU and O-DU entities over F1-c, according to the F1AP protocol (3GPP 38.473, Rel. 16 [87]), supporting UE end-to-end registration and PDU session establishment. Additionally, it supports user plane traffic over F1-u using GTP-U as per 3GPP 29.281, Rel. 16 [88].

Performance validation of the O-CU O-DU split using real RFs and Commercial Off The Shelf (COTS) UEs is ongoing, with interoperability testing initiated between the OAI O-DU and a commercial O-CU from Accelleran. Efforts are also underway to integrate the O-CU-UP/O-CU-CP split over the E1 interface and extend support for multiple O-CUs O-DUs.

Beyond core functionalities, OAI provides additional RAN features, supporting Software-Defined Radio platforms like USRP, BladeRF, and LimeSDR for integration across hardware environments. Integration with O-RAN Alliance interfaces, such as O1, is still in development. Multi-Radio Access Technology (Multi-RAT) support allows seamless coexistence between LTE and NR technologies.

OAI, written in C++, includes build scripts and dependencies that may pose vulnerabilities. Testing tools like RFs simulators and unit tests enhance network deployments, ensuring robustness. OAI offers scripts for deploying 4G Evolved Node Bs (eNBs), gNBs, and UE on various platforms, with containerized deployment via Docker and Kubernetes. The project includes tutorials, code guidelines, logging tools, and a Jenkins-supported continuous integration framework. The build process allows for verbose logging and tracing with the T tracer.

D. Simu5G

Simu5G is an open-source project created by the University of Pisa that evolved from the SimuLTE 4G network to support 5G NR based on the OMNeT++ framework [89]. This allows users to develop new modules that implement new algorithms and protocols, fostering high extensibility. The project originally targets the simulation; however, recently, they also provided a set of emulation modules capable of integrating with real 5G traffic [89]. The project can also run with eNBs and with the SimuLTE project. Its architecture is shown in Fig. 7d.

Real-time emulation with Simu5G is achieved using the OMNeT++ environment's event scheduler and the INET library for packet exchange between the simulation and the operating system. Lightweight modeling of Simu5G is crucial for real-time execution on desktop machines. New functionalities enable large-scale emulation of intercell interference and resource contention. Multiple models of key network elements, such as UEs and cells, balance detail and processing overhead. Some UEs are fully modeled to track application packets, while others are simplified to create resource contention. Similarly, some cells handle full protocol stack operations, while others focus on interference and MAC layer scheduling. The architecture features a BackgroundTrafficGenerator for simplified UEs, with Mobility and TrafficGenerator submodules for managing movement and data buffer size. The simplified cell module includes a BackgroundScheduler for resource block allocation and a BackgroundCellChannelModel for SINR computation, facilitating the deployment of multiple simplified cells with configurable parameters. This design optimizes resource management and interference modeling for efficient real-time emulation.

Simu5G is written in C++ and can be compiled with CMake, with g++ installed. The configuration of the emulated scenario can be performed with .ini files. The documentation is precise but very concise, also tests are not available. The project leverages the OMNeT++ framework and the INET library to exchange packets between a running simulation and the operating system, this significantly reduces the list of dependencies. Leveraging the OMNeT++ framework allows compatibility with both Windows and Ubuntu systems. Simu5G does not support any type of containerization nor management and orchestration platforms, also DevOps tools are not deployed to assist users in the extension of Simu5G or in the operations performed to deploy the tool. For logging Simu5G leverages the spdlog library to register the systems actions.

E. FikoRE

FikoRE (Framework for Immersive communication network Optimization RAN Emulator) is an open-source RAN emulator developed by Nokia [90] to lower entry barriers for application layer researchers. It enables testing of real-time solutions in realistic RAN deployments, serving as a rapid prototyping testbed for novel use cases. However, this focus limits its extensibility to other radio technologies like 4G. FikoRE incorporates simplifications from 3GPP specifications to prioritize scalability and performance, handling throughputs up to 1 Gb/s of actual IP traffic per transmission direction. The software architecture of FikoRE is depicted in Fig. 7e.

FikoRE is organized into three main modules: the Traffic Capture/Generator (TC/TG) module, the MAC Layer, and the UE Module. The TC/TG module generates virtual traffic based on user-selected models, parses IP traces to create realistic traffic, and captures actual IP traffic from specified ports. The MAC Layer handles resource allocation, adhering to 3GPP 38.211 [91], 38.214 [92], and 38.306 [93] specifications. The UE Module models a single UE, incorporating position estimation, channel quality estimation, and packet handling while abstracting the RLC, PDCP, and PHY layers. The PHY layer models key channel metrics such as SINR, RSRP, and HARQ retransmissions, as described in 3GPP 38.901 [94]. These modules are implemented for both uplink and downlink transmission directions.

The emulator operates with a 1 ms time granularity, differentiating traffic into emulated and simulated types. Simulated IP packets are generated based on a user-selected model, while real packets are filtered and queued by Netfilter, which provides an API for managing queued packets. These packets are sent to the PDCP/RLC layer, where simulated packets with matching IDs and sizes are queued.

The PDCP/RLC layer communicates with the MAC layer to allocate resources based on channel quality metrics, segments packets, and passes them to the PHY layer for emulation, including latency and HARQ retransmissions. Retransmitted packets face delays and are discarded if retransmitted too many times. Successfully transmitted real packets are released by Netfilter after the specified latency. A logging module tracks throughput, latencies, channel quality, error rates, and resource allocation. The UE module simulates individual UEs, generating traffic from a model or incoming IP traffic, updating mobility, estimating Channel State Indicators, and managing packet releases or drops.

FikoRE, written in C++11, uses a multithreaded approach for scalable emulation, ensuring real-time requirements are met with a 1 ms deadline for packet transmission. A ticker module synchronizes the emulator, triggering a signal every millisecond. Two main processes launch threads: one for MAC resource allocation and another for UE handling, with MAC Layer processes prioritized. Users can run the emulator in a single thread for debugging, but this is not recommended for real-time operation. The project emphasizes modularity for easy modifications to resource allocation algorithms. Deployment involves compiling with a Makefile using g++ on Ubuntu, requiring dependencies like Netfilter Queues and Minimalistic Netlink. Emulation parameters are set through a configuration file with a .ini extension, but dynamic reconfiguration is not supported. The authors provide tested configurations and well-written documentation. FikoRE does not support containerization or DevOps tools; instead, it focuses on providing a plug-and-play emulation tool for researchers.

F. Lessons Learned

The evaluated RAN projects offer open-source solutions for RAN emulation or RAN real testbeds. O-RAN SC presents a

TABLE III: Comparative table of the emulation and real testbed solutions

	Feature	O-RAN SC [50]	SRSRAN [78]	OAI [79]	Simu5G (Emulator) [80]	FikoRe [90]
General	**Standard Compliance**	3GPP, O-RAN Alliance	3GPP, O-RAN Alliance	3GPP, O-RAN Alliance	3GPP compliance at L3, L4	No, several simplification from 3GPP
	Type	Emulation, Real Testbed	Emulation, Real Testbed	Emulation, Real Testbed	Emulation, also Real-Time traffic	Emulation, also Real-Time traffic
User Plane	**UE**	None	Yes, connection with srsUE from 4G project and COTS UE	Yes, connection with COTS UEs, UEs	Yes, connection with emulated and COTS UEs	Yes, connection with real-time/simulated traffic UEs
	Physical Layer	Based in Intel FlexRAN. Implements 3GPP 38.211, 212, 213, 214, 215	Leverages UHD Hardware driver. MATLAB testvectors.	Implements 3GPP 36.211, 36.212 , 36.213, 38.211, 38.212	Implements: 3GPP RP18.0524	Implements 3GPP 38.901
	Data Link Layer	MAC:DPDK, RLC, PDCP SDAP. Everything 3GPP compliant	MAC:DPDK, RLC, PDCP, SDAP. Everything 3GPP compliant	MAC, RLC 38.322, PDCP 38.323, SDAP 37.324.	MAC, RLC, PDCP, IP2NIC inside NRNIC non compliant	MAC 3GPP compliant , RLC, PDCP non compliant
	Network Layer	Not in the project scope	RRC 3GPP compliant	RRC 3GPP 38.331 compliant	IP Traffic	IP traffic
	RF simulation	No	For testing purposes with ZeroMQ	Yes, with ZeroMQ	Yes	Yes
	4G EPC integration	No	Yes	Yes	No	No
	Compatible CN	Not tested, missing interfaces	Open5GS, Free5GC	Open5GS, Free5GC	Not tested	Not tested
	Split Support	7.2	7.2, 8.0	7.2	None	None
Control Plane	**Split Units**	DU-High , DU-Low, CU project halted, old release available	RU, and USRP tested, DU-Low, DU-High, CU-CP, CU-UP	RU, and USRP tested, DU-Low, DU-High, CU-UP, CU-CP	Not in the project scope	Not in the project scope
	O-RAN Interfaces	E2, O1	E2	E2, O1 (by December 2024)	Not in the project scope	Not in the project scope
	Multi-RAT Support	No	Yes, LTE, 5G NR	Yes, LTE, 5G NR	Yes, SimuLTE and 4G eNBs	No
DevOps	**Plan: Code planning, collaboration, version**	Confluence, Gerrit, GitHub	None, GitHub Discussion, GitHub	None, None, GitLab	None, GitHub discussion, GitHub	None, None, GitHub
	Code: Main language	C, C++	C++, Pyhton	C, C++	C++	C++
	Build: dependencies	13 dependencies	8 dependencies	22 dependencies	2 dependencies	2 dependencies
	Build: compiler	CMake (DU-Low: ICC, DU-High: gcc)	CMake (gcc, Clang)	CMake (gcc)	CMake (gcc)	CMake (g++)
	Test: code testing	Unit test available	Unit test available	Unit tests available	Not available	Not available
	Release: CI/CD tools	Jenkins	GitLab CI	Jenkins	None	None
	Deploy: Reference OS	Ubuntu (vN.A)	Ubuntu (v22.04), Fedora, Arch Linux	Ubuntu (v14.04-22.04), Windows 10 for UEs	Ubuntu (v20.04), Windows	Ubuntu (vN.A.), Windows
	Deploy: Containerization	Yes, Dockerfile	Yes, DockerFile	Yes, DockerFile	No	No
	Deploy: types	Bare Metal, Kubernetes	Bare Metal, Kubernetes, Helm	Bare Metal, Docker Compose, OpenShift	Bare metal	Bare Metal
	Operation: Configurations	Using .cfg files, .xml for PHY layer	With YAML files	Using build options and	OMNet .ini files	With .ini files
	Monitor: tools	Runtime logs	Runtime logs	Logs enabled at build time Traces with T tracer	spdlog library logs	Runtime logs

partial implementation of the control plane; in particular, it focuses on implementing the O-DU High, O-DU Low, and O-CU. Its main advantage is full compliance with common DevOps tools and wide support for developers. However, the community is pushing towards partially integrating O-RAN SC with OAI, and parts of the project will be deprecated. srsRAN project presents a complete 3GPP compliant implementation of the RAN, with also the possibility of using RFs simulation. FikoRE and Simu5G present a data plane emulation solution with real-time traffic support but with limited integration with the other RAN components, as none of them provides E2 or O1 implementation. Additionally, Table III summarizes the characteristics of each project, while Fig. 7 compares projects based on their implementation. It is noteworthy that the open-source project Free5GRAN[6], which only implements the PHY layer and has been inactive since 2021, is not included in this analysis. Similarly, OAIC [13] is excluded due to its reliance on an outdated version of srsRAN for the RAN implementation, which overlaps with the projects under review, thus contributing minimally to the objectives of this paper.

[6]https://github.com/free5G/free5GRAN.git

VI. O-RAN Control Plane Components

This section overviews the control plane components of the O-RAN architecture. In particular, we first describe the Near-RT RIC components and implementations used in the simulation and the emulation. Finally, we briefly describe the components and correspondent implementation of the Non-RT RIC and SMO components.

A. Near-RT RIC

The Near-RT RIC is a key element in the O-RAN architecture, serving as a mediator between the RAN and xApps. In the remainder of this subsection, we analyze the state of the art by presenting the main Near-RT RIC implementations and projects and describing their features. For each Near-RT RIC analyzed, its description includes, in this order, a quick description of the implementation project, including its contributors and a description of its main approach, its general architecture, as well as a description of each architectural component, its expected deployments using the platforms supported or recommended by its developers, and its advantages and disadvantages concerning other approaches.

TABLE IV: Comparative table of the Near-RT RICs: Elements marked with a † are compatible with the Near-RT RIC but must be configured and implemented for the xApps at the deployment time.

	Feature	**Embedded RIC** [45]	**ColO-RAN RIC** [25]	**FlexRIC** [44]	**μONOS RIC** [76]	**O-RAN SC RIC** [43]
General	**RIC architecture type**	Single artifact	Multiple components	Single artifact	Multiple components	Multiple components
	Modular architecture	No	Yes	No	Yes	Yes
	xApp coupling to Near-RT RIC	Yes	No	Yes	Partial	No
User Plane	**Supported service models**	KPM†, NI†, RC†, CC†, non-standard service models†	KPM†, NI†, RC†, CC†, non-standard service models†	KPM with Protocol Buffers encoding (ver. 2.01, ver. 2.03, ver. 3.00), RC with Protocol Buffers encoding (ver. 1.03), FlexRIC's non-standard service models with Protocol Buffers encoding	KPM (ver. 2.0.3 with Protocol Buffers and ASN.1 encoding), RC with ASN.1 encoding, μONOS non-standard service models with ASN.1 encoding	KPM†, NI†, RC†, CC†, non-standard service models†
	SM coupling with near-RT RIC	No	No	Yes	Yes	No
Control Plane	**Fully compliant functionalities implemented**	E2 termination, all the rest†	E2 termination, Database, Shared Data Layer, messaging infrastructure, API enablement	E2 termination, Database, Shared Data Layer, messaging infrastructure, API enablement	E2 and A1 terminations, Database, Shared Data Layer, messaging infrastructure, API enablement, conflict mitigation, xApp repository, security	E2, A1, O1 terminations, Database, Shared Data Layer, messaging infrastructure, API enablement, conflict mitigation, xApp repository, xApp subscription management, management, AI/ML support, security
	Partially compliant functionalities implemented	All except E2 termination†	xApp subscription management, AI/ML support, management, xApp repository	xApp subscription management, AI/ML support, management, xApp repository	xApp subscription management, AI/ML support, management	None
	Non-implemented functionalities	All except E2 termination†	A1, O1, Y1 terminations, conflict mitigation, security	A1, O1, Y1 terminations, conflict mitigation, security	A1, O1, Y1 terminations	Y1 termination
DevOps	**Plan: Code planning, collaboration, version**	None, None, None	None, GitHub Discussion, GitHub	None, None, GitLab	None, GitHub discussion, GitHub	Confluence, Gerrit, GitHub
	Code: SDK-supported programming languages for xApp development	Any Turing-complete†	C, C++, Python	C, C++, Python	Python	C, C++, Python, Go, Rust
	Code: RIC API to xApps	None	Redis API	FlexRIC SDK	gRPC	Redis API
	Build: compiler	Language-dependent†	gcc, g++, CPython	CMake (gcc, g++, CLang), CPython	CPython	CMake (gcc, g++, CLang), CPython Go, rustc
	Test: RIC code testing	Unit test available†	Not available	Unit test available	Unit test available	Unit test available
	Release: RIC CI/CD tools	None	None	GitLab CI	GitHub Actions	GitHub Actions
	Deploy: Reference OS	Linux†, Windows†, macOS†	Ubuntu (v22.04)	Ubuntu (vN.A.)	Ubuntu (v22.04)	Ubuntu (v22.04)
	Deploy: xApp Containerization support	Yes†	Yes	Yes	Yes	Yes
	Deploy: xApp deployment options	Bare Metal†, Docker Compose†, Kubernetes†	Bare Metal, Docker†, Docker Compose†	Integrated component of container	Helm and Kubernetes Operator	Helm and Helm Operator
	Operate: xApp Config	None	Dockerfile editing	Build options	μONOS config CLI	O-RAN SC xApp descriptor
	Monitor: tools	Runtime logs	Runtime logs	Runtime logs	μONOS monitoring	Runtime logs

1) Embedded RIC: The first Near-RT RIC option we examine is the absence of a separate Near-RT RIC, which we refer to as an embedded RIC [45]. This approach was proposed by the Open AI Cellular consortium as part of their documentation on creating custom xApps [45]. In an embedded Near-RT RIC scenario, instead of a dedicated RIC element, each xApp is expected to perform Near-RT RIC functionalities as part of its regular operation. xApps with embedded Near-RT RICs are not enforced to have any concrete architecture, as they are not clients towards a given SDL API, and instead, they expose an E2 termination that gNBs must connect to in order to be controlled, as depicted in Fig. 8a. Moreover, in Embedded RIC scenarios, there is no dedicated module providing a SDL, E2 message management and routing, or any other of the Near-RT RIC services, and each xApp must implement all of these services along with its functionalities. Embedded RICs are not subject to restrictions in the platform they can use to be deployed and can thus be executed as traditional applications, containers, virtual machines, or managed by an orchestration system.

Using an Embedded RIC provides xApps great freedom, as they are not subject to a given SDL API, a set of supported

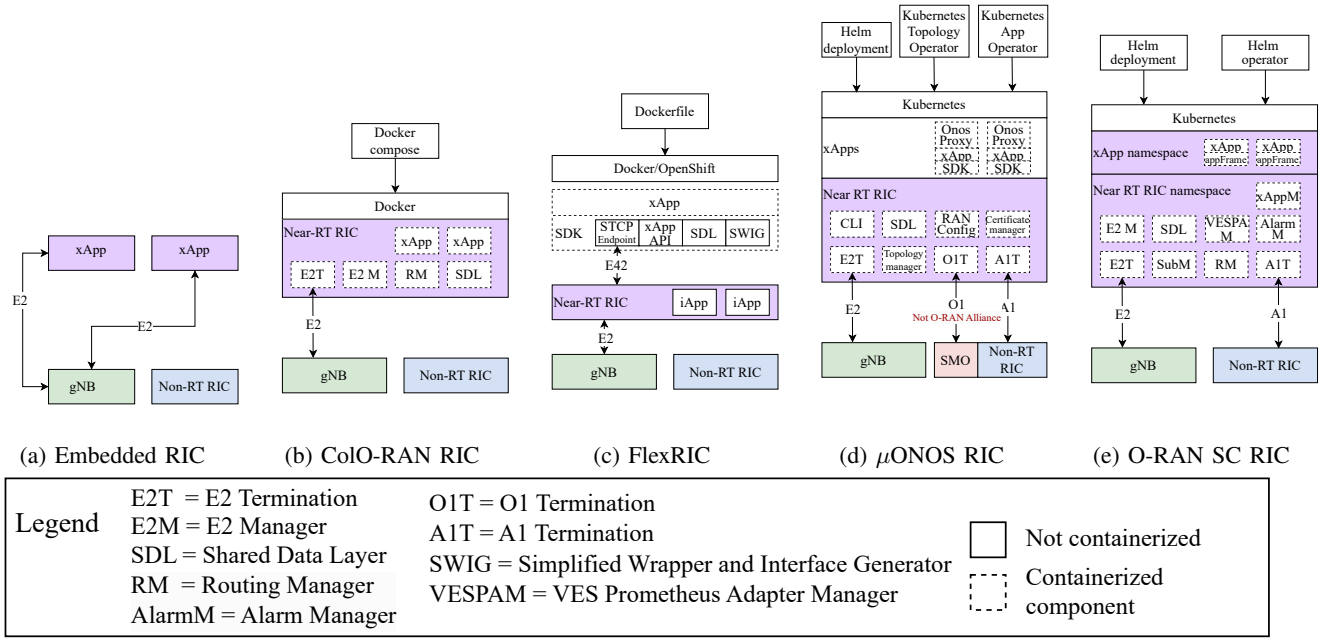


Fig. 8: visual architecture of each Near-RT RIC and xApp, including the involved software components, the target platform (if present), the deployment/configuration tool, and the interfaces with other RAN functions (if present).

service models, or a given platform. Moreover, the xApp may implement a minimal subset of the RIC services to be more efficient (e.g., an in-memory storage rather than a fully-fledged data layer). However, Embedded RICs do not scale well, as each additional xApp must replicate all the relevant Near-RT RIC functionalities and connect to all gNBs, converting the one-to-many connection achieved with a separate Near-RT RIC into a many-to-many connection that gNBs must manage, which is shown with the additional connections required in Fig. 8a, where each E2 termination in the RAN must be connected to the two E2 terminations of both xApps. Overall, Embedded RICs can be an option to implement functionalities unsupported by other existing Near-RT RICs, or to improve efficiency in use cases where a single xApp will be deployed.

2) ColO-RAN RIC: ColO-RAN RIC is the next Near-RT RIC we analyze, which was developed by forking the O-RAN SC RIC as a proof-of-concept Near-RT RIC for the ColO-RAN framework for the development of AI-based xApps [25]. Concretely, the ColO-RAN RIC is a fork from the *bronze* release of the O-RAN SC RIC, modified to ease the deployment and to remove some components, as well as altering the original logic of other elements to simplify some procedures. The ColO-RAN RIC is also known as the *Colosseum RIC*, *OpenRAN Gym RIC*, or *Northeastern RIC*, referencing its platform, project, and developer, respectively, although we refer to it as ColO-RAN RIC due to its original framework and purpose. The approach of this Near-RT RIC is to maintain a simpler version of the O-RAN SC RIC that provides the necessary functionalities. The defining architectural feature of the ColO-RAN RIC is its modularity. It consists of four modules (E2 Termination, E2 Manager, Routing Manager, Shared Data Layer) that can run independently, ensuring the ColO-RAN RIC remains operational as long as all four modules can communicate with each other. Moreover, xApps are also modular, only needing to communicate with one of the modules.

As shown in Fig. 8b, the first module is the E2 termination, which allows the gNBs in the RAN, aggregated or disaggregated, to communicate with a single endpoint. This is a common design feature of Near-RT RICs that addresses the complexity of E2 interface connections in Embedded RICs: a single Near-RT RIC, through a single E2 termination, can control an arbitrary number of gNBs, simplifying the setup and management of connections for xApps, the Near-RT RIC itself, and gNBs. The messages sent and received from the E2 interface are first handled by the Routing Manager module, which is concerned with steering the messages received from the RAN towards the appropriate xApps and the messages received from xApps or other Near-RT RIC components to the relevant E2 terminations in the RAN. The E2 Manager module handles the lifecycle of the E2 termination and stores key information received from the RAN (e.g., IDs of gNBs that have been registered in the Near-RT RIC) in the SDL. The E2 Manager is also the final handler of E2 messaging, as it collaborates with the Routing Manager module to send messages to the xApps through the SDL and send the messages xApps store in the SDL to the RAN. Finally, the SDL module serves two purposes: on the one hand, the storage of important Near-RT RIC information and, on the other hand, as a publish-subscribe messaging API for the xApps to interact

with the RAN. While the other three components are specific implementations for their purpose, the SDL uses Redis, and thus, the Redis API, to communicate with xApps.

The ColO-RAN RIC is implemented using Docker, providing the necessary files to build and run Docker containers. However, it does not include tools for managing or orchestrating these containers. xApps only need to communicate with the SDL container and can be deployed in various forms, such as other containers or standalone applications. The primary advantage of ColO-RAN RIC is its modularity and service decoupling, allowing for deployments in low-latency environments as it needs to respond in under a second. xApps can be independently developed and deployed, even on separate machines, without altering or restarting the Near-RT RIC. However, deploying the ColO-RAN RIC can be complex and resource-intensive for simple scenarios, requiring the deployment of 4 Docker containers and the management of their connections and discoverability, such as ensuring the E2 Manager has the correct IPs and ports for the E2 Termination and SDL.

3) FlexRIC: FlexRIC [44], developed by EURECOM as part of the Mosaic5G initiative, is a Near-RT RIC that focuses on minimizing the Near-RT RIC's overhead and delivering simplicity to xApp developers. FlexRIC introduces the term *Internal Application for Controller Specialization (iApp)* to the O-RAN ecosystem, as you can see from Fig. 8c. Whereas xApps are considered to be software artifacts, generally developed by third parties (i.e., different from the RIC developers) that leverage the Near-RT RIC's services, iApps are indivisible modules of the Near-RT RIC that provide additional, usually non-standard services. For example, the iApp currently included with FlexRIC performs validation and logging of E2 messages received from the RIC. Along with iApps, it is possible to develop and deploy custom xApps in FlexRIC. However, this xApps must be developed using the FlexRIC SDK, which includes different custom interfaces generated with SWIG for the Python, C, and C++ programming languages to interact with FlexRIC. Also, the SDK includes an STCP Endpoint to communicate with the rest of the Near-RT RIC, a set of xApp APIs, and a Shared Data Layer implemented with SQLite. In architectural terms, FlexRIC has a modular architecture that enables adding or removing components such as xApps or iApp, but its design enforces its deployment as a single monolithic block that encompasses all, Near-RT RIC services (e.g., E2 termination), iApps, xApps, and SDK modules. The main architectural module of the FlexRIC, aside from iApps, is the SDL, which FlexRIC internally calls the *RAN Node Information Base*, but we label as SDL as it is the name used in other RICs and the specification. As a single monolithic module, FlexRIC is distributed as a single Docker container, as well as in source code that can be built to create a traditional application.

Having to develop xApps using the FlexRIC SDK is the main limitation of this Near-RT RIC. This limitation on freedom, nonetheless, has its benefits: FlexRIC SDK gives xApp developers a simplified interface for development, as it automatically manages key aspects such as service models, encodings, or interaction with the SDL. It is noteworthy, however, that these xApps must be deployed in the same container or machine as FlexRIC to maintain a SDL across xApps and iApps, as well as to guarantee their communication. This is enforced for two reasons: first, the deployment in a single machine minimizes the overhead due to network communications; and second, FlexRIC's design considers the Near-RT RIC as a single entity that is specialized through xApps for a given purpose, rather than as a platform to enable interaction across xApps and the RAN.

4) μONOS RIC: μONOS RIC [76] is the Near-RT RIC introduced by the SD-RAN project. Proposed by the Open Network Foundation, the objective of the SD-RAN project is to provide a cloud-native Near-RT RIC based on the architecture used by the μONOS controller for software-defined networks [42], characterized by its disaggregation into different, independently deployable modules, which communicate using various protocols, such as gRPC, gNMI, or JSON/HTTP. Moreover, the μONOS RIC tries to automate and simplify the installation, uninstallation, deployment, and interaction with its elements by providing a CLI. In terms of architecture, the μONOS RIC is fully modular, based on small and highly cohesive *microservices*.

Figure 8d shows the architecture of the μONOS RIC, which comprises 8 modules. At the bottom, we find the E2 Termination, used to connect with the RAN similarly to ColO-RAN RIC or FlexRIC. Accompanying this termination, we have the A1 termination to connect the Near-RT RIC to the Non-RT RIC. The next module is the *ONOS operators*. Since Kubernetes orchestrates μONOS RIC, it provides custom Kubernetes Operators and resources that allow μONOS RIC modules, as well as xApps and other elements, to directly interact with the Kubernetes control plane. In Kubernetes, such custom definitions are known as *operators*. Concretely, there are two operators in μONOS RIC: the *app* operator, which automatically intercepts requests for deploying xApps, adding an *ONOS proxy* sidecar to them; and the *topology* operator, which defines custom topology-related resources, making Kubernetes aware of them so any topology changes are immediately notified to the μONOS RIC. These notifications arrive precisely at the *topology manager* module, inherited from the legacy μONOS RAN controller, whose role is to maintain up-to-date information about the topology elements with E2 Terminations (e.g., gNBs). UE information is stored in the SDL instead, then we find the *RAN configuration* module, which handles the subsystem for configuration management in the RIC. These modules can be interacted with by leveraging the *Command Line Interface (CLI)* module, as each of the modules and xApps define CLI commands for their interaction. Finally, the 8th module is the *ONOS proxy*: xApps for the μONOS RIC must be developed using the SDK for the Python and Go programming languages. Nonetheless, xApps are not part of the RIC. Instead, these SDKs abstract the API calls to the ONOS proxy, a separate process replicated as a separate pod for each xApp, becoming a *sidecar* for each. The ONOS proxy provides implementations of the KPM and RC service models, as well as non-standard service models.

The μONOS RIC is deployed using Helm charts, which are managed by Kubernetes and provided through a Makefile-

based CLI, which also provides commands to stop, uninstall, or manually interact with the μONOS RIC. Moreover, it is possible to deploy and maintain configurations with the Kubernetes Operators for the topology manager and the xApp components. In summary, μONOS RIC is a highly modular and Kubernetes-hosted Near-RT RIC, where xApps can be deployed independently of μONOS RIC, although not from its sidecar proxy.

5) O-RAN SC RIC: Finally, we analyze the Near-RT RIC proposed by the O-RAN Software Community, the O-RAN SC RIC [43]. In this tutorial, we analyze the recommended deployment of the *I release* of the O-RAN SC RIC, which is the latest stable version to date. The O-RAN SC RIC takes a similar approach to the ColO-RAN RIC by enabling a modular and disaggregated Near-RT RIC. However, while ColO-RAN emphasizes simplicity, the O-RAN SC RIC focuses on orchestrating its components and aims to serve as the reference Near-RT RIC according to the O-RAN specifications. Architecturally, the O-RAN SC RIC is built as a set of separate modules, each with a defined purpose and a concrete task within the Near-RT RIC, as shown in Fig. 8e.

The first component after the E2 termination is the *Routing Manager* component, functionally similar to the one in the ColO-RAN RIC by steering RAN messages to the appropriate xApps and vice versa. The *E2 manager* component is also similar to the one in the ColO-RAN RIC, with the task of handling the lifecycle of the E2 termination and connecting it with the SDL. Nonetheless, the management of the subscriptions, which is made directly by the components in the ColO-RAN RIC, is left to the *subscription manager* in the O-RAN SC RIC. This component serves as an intermediary between the xApp and the routing manager for messages related to subscriptions to RAN information, such as periodical KPI reports. xApps can subscribe to reports through a REST API provided by the Subscription Manager, which is registered as a distinct component in the Routing Manager component. Whenever a subscription message from the RAN arrives at the Near-RT RIC, the subscription manager submits a notification to the xApp. The main advantage of the Subscription Manager is that if multiple xApps subscribe to the same type of reports in the RAN, these subscription requests are not handled by the RAN, and instead, the subscription manager automatically multiplexes the reports to all xApps that subscribed to them, enhancing the efficiency of RAN subscriptions.

Then we find the *xApp Manager*, which provides a different view on xApp deployment compared to the other Near-RT RIC. To deploy or remove an xApp from the O-RAN SC RIC, it is necessary to *onboard* them, a process that consists of informing the Near-RT RIC of the xApp's presence or absence. This onboarding process is provided as a REST API by the xApp manager, enabling the deployment and removal of xApps, as well as for the query about the status and health of each xApp instance. Finally, in the same manner as the ColO-RAN RIC, we find the SDL, which is an instance of Redis. It is also possible to optionally deploy *A1* termination to enable connectivity from the O-RAN SC RIC to other Non-RT RIC. Finally, the Alarm Manager exposes alert when problems to the platform occur, while The VNF Event Streaming Prometheus Adapter (VESPAM) adapts internal data collection from the Near-RT RIC using Prometheus to scrape metrics from the platform and xApp microservices.

Regarding management of the component's deployment, the O-RAN SC RIC manages its components through Kubernetes, Helm, and the Helm Operator. By leveraging different configuration files, it is possible to select the components that will be instantiated and how. Moreover, xApps are also to be executed as Helm charts, although the O-RAN SC RIC provides supporting software to create and *onboard* Helm charts from the xApp's configuration files. Overall, the O-RAN SC RIC provides a similar environment to the ColO-RAN RIC, more complex, due to its higher number of components and its management through Kubernetes rather than through Docker containers.

6) Compatibility with simulation, emulation, and real testbed tools: In the context of this tutorial, it is crucial to analyze the compatibility of the available Near-RT RICs implementations with the simulation, emulation, and real testbed tools described in Sec. IV and in Sec. V. On the one hand, this analysis allows readers who have a specific interest in a target simulator (e.g. ns-3 experts), or emulator (e.g. OAI, srsRAN experts) to select a compatible Near-RT RIC and vice versa. On the other hand, it enables comparison across simulators, emulators, real testbeds, and Near-RT RICs based on the compatibility with other tools (e.g. which is the most compatible Near-RT RIC). A summary of this compatibility is provided in Table V for simulators and in Table VI for emulators and real testbeds, and the compatibilities and incompatibilities across different RAN tools and Near-RT RICs is detailed in the following. It is important to note that this compatibility analysis refers to both simulators, emulators, real testbeds, and Near-RT RICs as they are officially distributed, and the possibility of modifying the RAN tools, Near-RT RICs, or both to make them compatible is outside the scope of this tutorial.

a) Simulation tools: In Table V, the first two simulators analyzed are 5G-LENA and 5G Toolbox. These simulators act exclusively on the user plane, not offering an E2 termination for any Near-RT RIC to connect to. Hence, they are not compatible with any Near-RT RIC, and cannot be used for evaluating control plane components. The case of POSS is slightly more complex: while it does not offer any E2 termination, making it incompatible with an Embedded RIC, POSS is implemented as a xApp, which could, in theory, be compatible with Near-RT RICs by connecting through their SDL. However, POSS expects the Near-RT RIC to offer its SDL through an Apache Kafka API, which none of the Near-RT RICs implement, rendering it incompatible with them. gNB E2SM Emu exposes an E2 termination, through which it communicates using FlexRIC non-standard service models with a Protocol Buffers encoding. As a result, it is compatible with FlexRIC, as well as with Near-RT RICs that do not enforce a service model (ColO-RAN, O-RAN SC), but incompatible with μONOS-SD RIC, which enforces its own service models. The SD-RAN simulator was built with μONOS-SD RIC in mind, offering the data μONOS' topology operators expect. However, it is deeply coupled with μONOS-SD RIC's services, and other Near-RT RICs do not

have the necessary functionalities to support this simulator. As the SD-RAN simulator and gNB E2SM Emu are similar in scope, it can be seen as an alternative to gNB E2SM Emu for the μONOS-SD RIC. Finally, ns-O-RAN offers a multiplexed E2 interface that can send and receive messages with the standard KPM and RC service models, encoded in ASN.1. Only FlexRIC's latest version supports ASN.1 encoding and is compatible with ns-O-RAN. Moreover, it lacks the orchestration information required by μONOS-SD RIC. The other three RICs are, nonetheless, compatible with ns-O-RAN, as they do not enforce specific system models.

b) Emulation and real testbed tools: In Table V, Simu5G [80] and FikoRE [90] act exclusively on the user plane, not offering an E2 Termination for any Near-RT RIC to connect to. Hence, they are incompatible with any Near-RT RIC and cannot be used to evaluate control plane components. The O-RAN SC [50] project currently does not expose any E2 Termination on the O-CU side; the project was started in 2021; however, it was recently archived. Another effort of O-RAN SC is the implementation of an E2 simulator capable of simulating messages from the gNB, arriving at the Near-RT RIC. OAI [79] is compatible with all the projects implementing a Near-RT RIC. srsRAN [78] results not being compatible with the Colo-RAN RIC project, while it works with all the other existing Near-RT RIC projects.

TABLE V: Compatibility table across simulators and near-RT RICs

	Embedded RIC [45]	**ColO-RAN RIC [25]**	**FlexRIC [44]**	**μONOS-SD RIC [76]**	**O-RAN SC RIC [43]**
5G-LENA [70]	×	×	×	×	×
5G Toolbox [73]	×	×	×	×	×
POSS [74]	×	×	×	×	×
gNB E2SM Emu [75]	✓	✓	✓	×	✓
SD-RAN simulator [76]	×	×	×	✓	×
ns-O-RAN [77]	✓	✓	✓	×	✓

B. SMO and Non-RT RIC

The SMO in O-RAN Alliance architecture is a component responsible for the RAN domain management. The SMO follows the principles of an SBA architecture, and it mainly focuses on three functionalities: the Fault, configuration, accounting, performance and security (FCAPS) interface to O-RAN network functions, the Non-RT RIC for RAN optimization, and the O-Cloud Management, Orchestration, and Workflow Management. The SMO connects through different interfaces to internal and external services. The A1 Interface connects the Non-RT RIC in the SMO to the Near-RT RIC, while the O1 Interface is used by the SMO for the FCAPS support of the O-RAN Network Functions. In particular, the SMO connects through the Open Fronthaul to O-RUs for FCAPS support. Finally, the O2 Interface between the SMO and the O-Cloud enables the management of platform resources and workloads.

TABLE VI: Compatibility table across emulators/real testbed implementations and Near-RT RICs

	Embedded RIC [45]	**ColO-RAN RIC [25]**	**FlexRIC [44]**	**μONOS-SD RIC [76]**	**O-RAN SC RIC [43]**
OAI [79]	✓	✓	✓	✓	✓
O-RAN SC [50]	×	×	×	×	×
srsRAN [78]	✓	×	✓	✓	✓
Simu5G [80]	×	×	×	×	×
FikoRE [90]	×	×	×	×	×

The Non-RT RIC is connected with the Near-RT RIC through the O-RAN Alliance's A1 interface from which it can receive forwarded raw data coming from the gNBs or already aggregated data or computations from the deployed xApps. The Non-RT RIC connection to the SMO is not defined by the O-RAN Alliance standard, leaving the interface between the two to the developers. Moreover, it optimizes the RAN by providing policy-based guidance, by running ML model management, providing enrichment information to the Near-RT RIC functions, and supporting internal control loops. The rApps interacts with the Non-RT RIC framework via the R1 interface. This interface enables rApps to obtain information and trigger actions (e.g., policies, re-configuration) through the A1 interface-related services.

1) Non-RT RIC by O-RAN SC: Academic and industrial research along with the O-RAN specifications are focusing more on the definition of frameworks for the Near-RT RIC, leaving the SMO and the Non-RT RIC unexplored. At the moment of writing, the only available open-source solution implementing O-RAN Alliance specification-compliant Non-RT RIC is the O-RAN SC's implementation, currently at Release I [43]. The O-RAN SC Non-RT RIC encompasses a range of components and services that facilitate various functionalities essential for managing and optimizing the RAN depicted in Fig. 9.

The *rApp Manager* supports onboarding, instantiation, and lifecycle management of composite rApps. It includes a preliminary package demonstrating lifecycle management of rApps with and without microservices. Using Open Network Automation Platform (ONAP) ACM [7] models and managers, it showcases an extensible ACM 'Participant' approach for diverse rApp constituents. It integrates with the Data Management Engine for data inputs and outputs, covering registration, discovery, and access control, and with the Service Management Engine for service inputs and outputs, including gateway setup and service mesh configuration.

[7]https://docs.onap.org/projects/onap-policy-parent/en/latest/clamp/acm/acm-user-guide.html

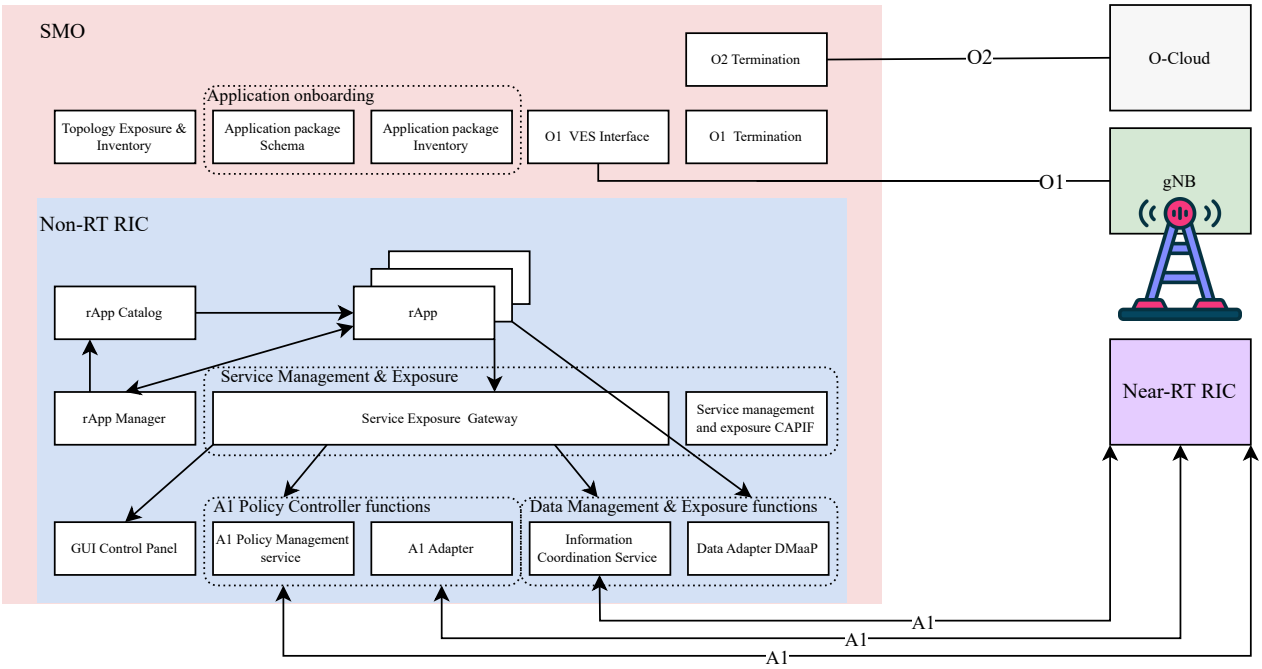


Fig. 9: O-RAN SC Non-RT RIC and SMO architecture referred to code release I

The Non-RT RIC *control panel* interface allows seamless interaction with Non-RT RIC services, enabling users to manage A1 policies in the RAN using a model-driven approach based on JSON schemas. It also handles producers and jobs for the Information Coordination Service and configures the A1 Policy Management Service, including Near-RT RIC management. Communication with the A1-Policy Management Service occurs via a REST-based Service Exposure gateway, and the interface is built using Angular [8] for a responsive user experience.

The Non-RT RIC *Service Gateway* is a key API gateway that enables applications and the Control Panel to access A1 services. Utilizing the Spring Cloud Gateway [9], it exposes the A1 Policy Management Service and the Information Coordinator Service. The gateway allows predicates to be added through code and YAML configurations, enhancing integration within the Non-RT RIC ecosystem. The Common API Framework (CAPIF) by 3GPP acts as a service registry for service management. The initial *rApp Catalogue* supports rApp registration and querying, with plans for integration into the rApp Manager in future releases. Additionally, the Non-RT RIC **Service API Gateway** enables applications to utilize Non-RT RIC, SMO, and other interfaces through a unified gateway. It supports dynamic registration and exposure of service interfaces, extending static gateway functions. The initial version, based on the Kong gateway [10], focuses on exposing A1 and O1 services for flexible integration.

The *A1 Policy Management Service*, based on ONAP CCSDK [11], is a microservice for managing A1 policies in a network. It maintains a transient repository of A1 policy instances, Near-RT RICs, and policy types, offering a unified REST API for operations like querying, creating, and updating policies. The service provides a persistent cache of A1 policy information, streamlining A1 traffic and supporting re-synchronization after inconsistencies or Near-RT RIC restarts. It accommodates multiple Near-RT RIC versions and can operate without the A1 Adapter, connecting directly to Near-RT RIC and supporting various Southbound connectors. Moreover, the *A1 Policy Controller/Adapter*, based on ONAP CCSDK, serves as an optional mediation point for A1 Policy interfaces within SMO and Non-RT RIC environments. It supports A1 REST southbound communication and RESTCONF northbound interactions, including adapters for protocol conversion. This controller/adapter provides mapping logic and provisioning capabilities, ensuring flexibility and enhancing interoperability across various network configurations.

The *Information Coordinator Service (ICS)* is essential for managing Information Jobs, producers, and consumers, functioning as an A1 Enrichment Information Controller. ICS acts as a data subscription service, allowing consumers to create subscriptions without knowing their producers' details. Each producer can generate various data types, which may be produced by multiple sources. Data consumers, including rApps via R1 APIs or Near-RT RIC through the A1 API, receive subscribed data known as Enrichment Information. ICS supports multiple subscriptions per consumer, each with configurable parameters.

[8]https://v17.angular.io/start
[9]https://spring.io/projects/spring-cloud
[10]https://docs.konghq.com/gateway/latest/
[11]https://docs.onap.org/projects/onap-ccsdk-oran/en/latest/guide/developer-guide.html

The *DMaaP/Kafka Information Producer Adapters* are configurable mediators that transform data from DMaaP [12] and Kafka [13] into coordinated Information Producers. They provide two implementations that allow Information Consumers to access DMaaP or Kafka events as coordinated Information Jobs, acting as producers for the ICS. The Java Spring [14] implementation enables filtered mediation for efficient data integration.

The *A1 Interface Simulator* serves as a stateful, extensible test stub for simulating Near-RT RIC with multiple A1 providers. Implemented as a Python application, it supports A1-Policy and A1-Enrichment Information, featuring a Swagger-based northbound interface for easy modifications of A1 profiles, including versions and policy types. Compatible with all A1 application protocol versions, it also includes a call-out feature for integrating specific A1-Policy behaviors via REST and Kafka, enhancing flexibility in testing and development.

The *RAN Performance Monitoring Functions* include tools for file-based and event-based RAN performance monitoring, covering data collection, parsing, filtering, storage, and forwarding. This comprehensive suite adheres to 3GPP standards (TS 32.432 [95] and TS 32.435 [96]) for data representation. Moreover, it allows subscribers, like rApps, to subscribe to specific measurement types. Key components are the Data File Collector for retrieving 3GPP-compliant XML files, and the File Converter, which transforms XML to JSON while maintaining fidelity. Performance-monitoring Producer instances serve as Information Coordinator Service Producers, facilitating data delivery. Additionally, the InfluxDB [15] Logger stores selected measurements in a time series database for efficient management and analysis.

Authentication is managed by a utility service that retrieves tokens from Keycloak, streamlining the process for services. Additionally, the initial Kubernetes Helm Chart lifecycle manager supports basic operations for rApp microservices, enabling onboarding, starting, stopping, and modifying services as Helm charts.

2) SMO by O-RAN SC: The SMO project by ORAN SC, shown in Fig. 9, requires essential components such as a Topology and Inventory Service, an Application Onboarding Service, O1 termination, and O1/VES for monitoring. The O2 interface is not yet implemented, but is expected to be added in future releases.

The SMO is under development, with its software components mapped to existing open-source projects. The *Topology Exposure and Inventory* manages network resources using evolving, vendor-agnostic data models for a unified view of network topology. This autonomously updated data, derived from inventory and configuration sources, illustrates network entities and their relationships for specific use cases. Topology and Inventory objects are standardized using YANG models.

The *O1 interface* utilizes the NETCONF protocol for managing O-RAN network elements like Near-RT RIC, O-CU, O-DU, and O-RU. The SMO functions as a NETCONF client, while the network elements serve as NETCONF servers. It employs OpenDaylight data models for configuration, with the OpenDaylight data models Community GUI for the user interface. Additionally, the SMO provides configuration REST APIs.

A key function of the SMO is *application onboarding* for both rApps on Non-RT RIC and xApps on Near-RT RIC. The SMO maintains an application package catalog for operators to deploy or create instances of these applications. It also aims to define an Application Package Schema based on ETSI NFV SOL 004 schema definitions for Virtualized Network Function Descriptors using TOSCA [16] and YANG data models. The *O1/VES* interface enables monitoring within SMO. Network functions connected to O1, such as Near-RT RIC, O-CU-CP, O-CU-UP, O-DU, and O-RU, send information to the SMO. Monitoring data is collected by a platform in the SMO, where VNF Event Streaming (VES) Agents format events into VES Events for the VES Collector. The Collector stores events in InfluxDB, and optionally in Elasticsearch or Kafka, making the data accessible for analysis with tools like Grafana.

3) Compatibility with other O-RAN tools: The compatibility of the SMO with gNBs depends on the O1 interface, which serves as a management interface for NFs to authenticate and configure. Existing open solutions for simulation, emulation, and real testbeds implementing O-CUs, O-DUs, and O-RUs should advertise their capabilities over a NETCONF session established between the SMO and each NF. This NETCONF session facilitates the configuration and management of NFs. Notably, this capability is available for OAI [17] and the O-RAN SC O-DU high project [18]. Due to the limited O1 implementations, O-RAN SC provides an O1 simulator [19] to test the SMO functionalities.

For SMO compatibility with O-Cloud, efforts are underway to implement the O2 interface for infrastructure management via defined APIs to retrieve inventory and monitoring information. The O-RAN SC implementation of the O2 interface exists in two repositories. The O2 implementation in the SMO wraps OpenStack Tacker [20] for Virtualized Network Function (VNF) management and Network Function Virtualization (NFV) orchestration [21]. The Platform interface project [22] integrates into the O-Cloud platform, providing infrastructure state information to the SMO. It includes two main services: the Infrastructure Management Service and the Deployment Management Service, which expose the O-Cloud infrastructure to the SMO. Current platform support is limited to OKD and StarlingX, with trials conducted using the O-DU and O-CU projects by O-RAN SC.

[12]https://docs.onap.org/projects/onap-dmaap-datarouter/en/latest/apis/data-router-api.html

[13]https://kafka.apache.org/24/documentation.html

[14]https://spring.io/projects/spring-framework

[15]https://docs.influxdata.com/

[16]https://docs.onap.org/projects/onap-vnfrqts-requirements/en/latest/Chapter5/Tosca/index.html

[17]https://gitlab.eurecom.fr/oai/o1-adapter

[18]https://docs.o-ran-sc.org/projects/o-ran-sc-o-du-l2/en/latest/overview.html

[19]https://github.com/o-ran-sc/sim-o1-interface.git

[20]https://docs.openstack.org/tacker/latest/

[21]https://github.com/o-ran-sc/smo-o2.git

[22]https://github.com/o-ran-sc/pti-o2.git

The A1 interface connects the Non-RT RIC with the Near-RT RIC. Three projects implementing the A1 interface for the Near-RT RIC, along with the A1 policy manager for the Non-RT RIC and an A1 simulator, are available on the O-RAN SC page. The xApp A1 mediator [23] operates in the Near-RT RIC, exposing a generic REST API for xApps to send and receive northbound messages. It validates the payload and communicates it to the xApp via RIC Message Router (RMR) messaging. The A1 Policy Management Service [24] in the Non-RT RIC provides a REST API for managing policies in the O-RAN architecture, maintaining a transient repository of all A1 policy instances, and tracking Near-RT RIC instances and supported policy types. The A1 Simulator [25] simulates the A1 as a generic REST API, validating payloads and applying policies to mimic an existing Near-RT RIC instance of the Non-RT RIC.

C. Lessons Learned

The O-RAN architecture presents a complex landscape with diverse approaches to implementing its control plane components. Notably, Near-RT RIC implementations vary significantly, from embedded solutions offering flexibility for single-xApps deployments to modular designs enhancing scalability and maintainability. Embedded Near-RT RICs, while offering freedom, struggles with scalability and violates software principles like disaggregation, single responsibility components, modularity, and xApp software personalization. Modular Near-RT RICs, such as ColO-RAN Near-RT RIC and O-RAN SC Near-RT RIC, promote service decoupling, enabling independent xApp development and deployment. However, compatibility challenges persist, as different Near-RT RICs enforce varying SDKs, APIs, and service models, hindering interoperability with simulation and emulation tools. For this reason, there are emerging xApp frameworks for streamilining xApp development in O-RAN SC [97]. Deployment and orchestration methods, like Docker and Kubernetes, also differ, impacting complexity. xApp development and management are handled differently across Near-RT RIC implementations, with some using sidecar proxies and others relying on SDKs or onboarding processes. The Non-RT RIC and SMO, responsible for overall RAN management and control, remain relatively unexplored compared to the Near-RT RIC, with O-RAN SC implementation being the primary open-source solution. These components focus on policy management, service coordination, and application lifecycle management, utilizing service-based architectures and API gateways. An effective application-level O-RAN project should abstract this complexity by simplifying component interactions and address compatibility challenges with the various simulated and emulated gNBs through standardization and open-source collaboration.

VII. Hands-on: from Simulation to Real Testbed

This section presents a hands-on tutorial that guides the reader through setting up a complete RAN environment. We will specifically focus on deploying a Near-RT RIC and an example xApp, along with a simulated and emulated gNB. Moreover, we will deploy a SMO and Non-RT RIC blueprint with simulated A1 and O1 interface. For a detailed description of Near-RT RIC, xApp, gNBs, SMO, and Non-RT RIC, as well as their individual roles in the RAN and the interfaces across them, refer to Section III.

This tutorial serves as a starting point for readers to learn about the development of xApps and rApps, as well as their testing, evaluation, and preparation for a real environment. Section VII-A presents the setup details, including the code, infrastructure, and projects used in this example. Section VII-B explains how to establish the simulation environment, Section VII-C covers the setup and evaluation of the xApp in a real environment, while Section VII-D presents a first SMO and Non-RT RIC deployment connected to simulated interfaces using both Kubernetes and Docker.

A. Hands-on setup

This hands-on will leverage some of the tools analyzed throughout the paper, both to align with the current tools and systems used by the O-RAN research community and to allow the reader to become familiar with some of the tools that have been analyzed. Specifically, six elements must be set up in this hands-on: the O-RAN SC xApp that will be evaluated, the O-RAN SC Near-RT RIC used by this xApp, the ns-O-RAN simulator that provides the simulated RAN environment, the OAI real testbed implementation over which the xApp will be finally executed, the O-RAN SC rApp, the Non-RT RIC and the SMO.

Note to the readers: This tutorial focuses on the development and migration of xApps from a simulated environment to a testbed, rather than on ML-based systems. The approach benefits from using an xApp with simple logic, similar to a *hello world* example in programming. This makes it accessible to non-ML-expert readers, as they can easily understand the xApp. Meanwhile, ML experts and other xApp developers can quickly locate the section of the code where they can replace the logic with their own model or custom functionality. To demonstrate this, we use the example xApp provided by the selected RIC, which prints the messages received from the RAN. This example is widely used and well-documented, making it ideal for illustration. Similarly, the rApp, a basic hello-world application for the Non-RT RIC, follows the same concept.

To function properly, the xApp must be connected to a Near-RT RIC. Out of the analyzed Near-RT RICs, the most compatible are the embedded RIC, the ColO-RAN RIC, and the O-RAN SC RIC. Using an embedded RIC in this example may be counterintuitive, as a key aspect of the hands-on experience is connecting an xApp to a Near-RT RIC, which the embedded RIC would overlook. Among the two remaining options, the ColO-RAN RIC is a good choice due to its simplicity and focus on simulation testing. However, the O-RAN SC RIC, an updated version of the same Near-RT RIC, better aligns with the tutorial objectives, as it is designed for use in more production-oriented environments. Therefore,

[23] https://github.com/o-ran-sc/ric-plt-a1.git
[24] https://github.com/o-ran-sc/nonrtric-plt-a1policymanagementservice.git
[25] https://github.com/o-ran-sc/sim-a1-interface.git

in this hands-on, we leverage the O-RAN SC RIC as our Near-RT RIC, as it is the most compatible Near-RT RIC intended for its use in real testbeds. Similarly, analyzing the simulators, gNB E2SM Emu is the most compatible, but this simulator is intended to perform integration testing and development, not to simulate a realistic RAN deployment. Instead, ns-O-RAN is best suited for this purpose, being also compatible with the O-RAN SC RIC but allowing for the simulation of the full user plane. Moreover, if the embedded or ColO-RAN RICs were to be used, ns-O-RAN would also be fully compatible. In terms of emulation implementation, we leverage the OAI emulated RAN, Core, and UEs, connecting to the O-RAN SC Near-RT RIC. This choice is endorsed for the completeness of OAI solution compared to other open-source solutions like srsRAN as highlighted in section V. Since O-RAN SC provides the only available implementation of the Non-RT RIC, we have chosen to present an initial Helm chart deployment in Kubernetes to align with the platform adopted by the above projects.

To set up the hands-on environment, it is required a machine with root privileges running `Ubuntu 22.04` OS and the `5.15.0-101-generic` kernel. The machine must have at least 6 CPU cores, 32 GB of RAM, and 100 GB of storage. The projects needs the following software in this environment: `git`, `make` version `1.4`, `kubectl` version `1.29.3`, Docker version `28.0.0` and `helm` version `3.14.3`. Also, a Kubernetes cluster should be set up to deploy the environment. To simplify this process, it is possible to create a Kubernetes cluster in a container using `kind` version `0.22.0 go1.20.13 linux/amd64`, which requires `Docker` version `24.0.5`. Although different software or versions may work, using the specified versions is recommended.

B. Simulation environment

For the simulation hands-on, we will deploy the O-RAN-SC Near-RT RIC, an ML-based xApp, and connect it to the ns-O-RAN simulator. Within the chosen software directory, we perform the O-RAN SC RIC in two steps. For the first step, whose code is shown in Listing 1, it is necessary to clone the code from the provided repository (lines 1-2) [26]. This repository includes the code for the O-RAN SC RIC's I-release, based on commit `f73a666` for the main repository. After the repository is cloned, it is necessary to create a new Kubernetes cluster, which we do using Kind (line 3). Within this cluster, we then create two namespaces: `ricplt` (line 4), which will host the Near-RT RIC elements, and `ricxapp` (line 5), meant for xApps. After the cluster and the namespaces are configured, `chartmuseum` version `0.13.1 (build 79bb39c)` is necessary. The repository includes it, and thus, it is enough to copy the binary (line 6). `chartmuseum` is installed to be used as a repository for Helm charts, which will then be pushed to it. Thus, the necessary Helm plugin must also be installed (line 7). It is then necessary to create the directory to store Helm charts in `chartmuseum` (line 8). Then, `chartmuseum` is launched on port `6873`, configured to use local storage on the specified directory (line 9). It is important to note that the `chartmuseum` process is executed in the foreground and thus takes control of the terminal. While advanced users may execute this process in the background, detached from their terminal, or in a terminal multiplexer, we recommend newer users to follow the rest of the setup in a separate terminal window or tab, which we highlight by separating into a different listing. Finally, to simplify this process, the repository already includes the code of Listing 1 as scripts: lines 3-5 are included as `setup-cluster.sh`, and `chartmuseum.sh` is based on lines 6-9.

```
1  git clone https://gitlab.com/MMw_Unibo/o-ran/o-ran-
       ↪ tutorial-hands-on
2  cd o-ran-tutorial-hands-on
3  sudo kind create cluster
4  kubectl create ns ricplt
5  kubectl create ns ricxapp
6  sudo cp chartmuseum-installer/chartmuseum /usr/bin/
       ↪ chartmuseum
7  helm plugin install https://github.com/chartmuseum/
       ↪ helm-push
8  mkdir $HOME/helm/chartsmuseum
9  chartmuseum --debug --port 6873 --storage local --
       ↪ storage-local-rootdir $HOME/helm/chartsmuseum
10
11 bin/deploy-nonrtric -f nonrtric/RECIPE_EXAMPLE/
       ↪ example_recipe.yaml
```

Listing 1: Near-RT RIC environment setup

The next steps, shown in Listing 2, involve instantiating the Near-RT RIC. To proceed, run the following commands in a new terminal while keeping the chart museum running, or use a `tmux` session. The Near-RT RIC installation process by O-RAN SC utilizes a Helm chart-based installer to deploy its platform components. The installation is configured via the `values.yaml` file, which can be customized using an override file. First, it is necessary to add two Helm repositories: the local `chartmuseum` instance and the InfluxDB repository, which the O-RAN SC RIC uses for some components (lines 1-2). The mandatory components such as `appmgr`, `dbaas`, `e2mgr`, `e2term`, `submgr`, and `rtmgr` are deployed by default, with optional components like `a1mediator` and `influxdb2` configurable through override files. These are all the components included in the `minimal-nearrt-ric.yaml`. Then, through the corresponding `make` recipe, the O-RAN SC RIC charts can be added to the local repository (lines 3-4). These first 4 lines are provided as a script in the repository (`make-charts.sh`), although it may be necessary to navigate to the directory (line 3) after executing it. Then, line 5 must be executed. If the response is `No results found`, it is necessary to re-run line 4 until the search finds the newly created chart. Next, line 6 installs the chart, assuming you're within the previously mentioned directory.

Using line 7 right afterward, it is possible to see the status of all the Kubernetes pods deployed. Before continuing the tutorial, it is necessary to wait until all the pods are in the `Running` status. It is noteworthy that, during the process, pods may crash, restart, fail, and enter `CrashLoopBackOff` status, this is the expected behavior of the RIC, and the reader should simply wait until they are eventually all running which, depending on the machine the RIC is executing on,

[26]https://gitlab.com/MMw_Unibo/o-ran/o-ran-tutorial-hands-on

can take between 2 and 15 minutes. Once all pods are running correctly, the O-RAN SC RIC can be considered to be correctly instantiated.

```
#run the following in a new terminal
helm repo add local http://localhost:6873
helm repo add influxdata https://helm.influxdata.com
cd ric-plt-ric-dep/new-installer/helm/charts/
make nearrtric
helm search repo local/nearrtric
helm install nearrtric -n ricplt local/nearrtric -f
    ↪ ../../helm-overrides/nearrtric/minimal-nearrt
    ↪ -ric.yaml
kubectl get pods -n ricplt
```

Listing 2: Near-RT RIC setup

Once the Near-RT RIC is running correctly, it is necessary to set up the xApp that will execute for this example. As mentioned, we use the `ric-app-hw` developed by O-RAN SC[27] as a demonstration, based on the `e-release` branch commit `308a280`. To onboard it, we use the provided `xapp_onboarder` component of the xApp manager in O-RAN SC[28], on commit `361faac`. Assuming the terminal is still in the last directory from Listing 2, the first line in Listing 3 involves moving to the directory of the onboarder. Then, the onboarder requires Python 3.9, which is set up in a virtual environment to ensure the cleanliness of the default environment used in the machine. This virtual environment is set up in lines 2-5. From then on, by activating the virtual environment (line 6), the reader will have access to Python 3.9. In this environment, we will install the requirements for the onboarder (line 7) and the onboarder itself (line 8). The repository provides a script that automatically encompasses lines 2-8 and deactivates the environment (`setup-python.sh`), which can be used together with line 6 to automatically get the virtual environment set up and activated. Following this setup, the environment will have the `dms_cli` command necessary to onboard xApps. Thus, using the command in line 9, the xApp is onboarded, while line 10 generates the necessary Helm chart, installed in the `ricxapp` namespace in line 11, all three included as `onboard.sh`. Finally, as the chart is installed, line 12 checks that the pod is correctly running. With this, the xApp is instantiated and running.

```
cd ../../../../ric-plt-appmgr/xapp_orchestrater/dev/
    ↪ xapp_onboarder
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt update
sudo apt install python3.9-dev python3.9-venv
python3.9 -m venv .venv
. .venv/bin/activate
python -m pip install -r requirements.txt
python -m pip install .
CHART_REPO_URL=http://localhost:6873 dms_cli onboard
    ↪  --config-file-path ric-app-hw/init/config-
    ↪ file.json --shcema_file_path ric-app-hw/init/
    ↪ schema.json
CHART_REPO_URL=http://localhost:6873 dms_cli
    ↪ download_helm_chart hwxapp 1.0.0
helm install hwxapp -n ricxapp hwxapp-1.0.0.tgz
deactivate
kubectl get pods -n ricxapp
```

Listing 3: xApp setup

[27]https://github.com/o-ran-sc/ric-app-hw
[28]https://github.com/o-ran-sc/ric-plt-appmgr

Finally, we install the ns-O-RAN image created for this tutorial, stored in `ns-o-ran-helm` (line 1). To build the ns-O-RAN Docker image, it is necessary to specify the IP address of the E2 termination in the O-RAN SC RIC (line 2). This is required because the gNBs needs to know the correct E2 termination to connect to. Once the IP is known, the Docker image provides a build argument for this IP and can be built with the correct IP using the command from line 3. These last two lines are provided as `build.sh` in the `ns-o-ran-helm` directory. If the image is correctly built, the next step is loading it into the registry of the Kubernetes cluster. In the case of the Kind cluster, this is done on line 4. Finally, we installed the Helm chart for ns-O-RAN on line 5.

```
cd ../../../../ns-o-ran-helm
e2termip=$(kubectl get endpoints service-ricplt-
    ↪ e2term-sctp-alpha -n ricplt -o jsonpath='{.
    ↪ subsets[0].addresses[0].ip}')
docker build -t ns-o-ran-mod:latest docker --build-
    ↪ arg E2_TERM_IP=$e2termip
kind load docker-image ns-o-ran-mod:latest
helm install ns-o-ran helm/
```

Listing 4: Simulator setup

The ns-O-RAN image in this hands-on executes the *scenario zero* that comes with the source code. This scenario was chosen due to its use in the original ns-O-RAN proposal [77], making it well-known for readers familiar with ns-O-RAN. This scenario involves 4 gNBs and an eNB, with one gNB co-located with an eNB at the center, and the remaining three gNBs positioned within a 1000-meter radius from the center. Additionally, 12 UEs are simulated, randomly distributed within the circle formed by the central gNB and the others. Each UE moves with a bi-dimensional random walk at a uniform speed between 2 and 4 m/s, simulating UDP traffic [77].

To check the environment is working properly, you can look at the logs of the E2 termination using a Kubernetes client (e.g., `kubectl get logs <pod-name> -n <namespace>`. In this specific instance, the E2 termination is in the `ricplt` namespace, while the specific name for the pod is decided at runtime and must be found out using `kubectl get pods -n ricplt`). You will find that the simulated gNBs have connected correctly to the Near-RT RIC, as announced by log lines at the `DEBUG` level with messages similar to those from Listing 5.

```
Accepted connection on descriptor 20 (host
    ↪ =::3134:3031:3331:3931%909195060, port=38472)
Accepted connection on descriptor 21 (host
    ↪ =::3134:3031:3331:3931%909195060, port=38473)
Accepted connection on descriptor 22 (host
    ↪ =::3134:3031:3331:3931%909195060, port=38474)
Accepted connection on descriptor 23 (host
    ↪ =::3134:3031:3331:3931%909195060, port=38475)
```

Listing 5: Selected E2 termination logs for simulated gnb connection

C. Real environment

The general installation prerequisite of the real testbed involves the installation of the O-RAN SC RIC as provided in

Section VII-B, specifically Listings 1 and 2. The installation is based on the same repository, which contains OAI Core v2.0.1 (commit `eb11579`), and the develop branch of OAI RAN (commit `0a603a10`). Moreover, `gcc-9` is required to build OAI gNB. To establish a connection between the OAI gNB and the O-RAN SC Near-RT RIC, follow these steps, ensuring the provided code block is used as a guide. Note that in this example, the RF is simulated. Initially, create the necessary Kubernetes namespaces to organize the deployment as in Listing 6.

```
kubectl create ns oai-core
kubectl create ns oai-ran
```

Listing 6: Kubernetes namespace creation

To build the OAI Core inside a Kubernetes cluster, follow the provided guide[29]

and note that you should not build the image of the single components from scratch, as the helm charts already use the `oaisoftwarealliance` repositories. The deployment has been tested with Calico as CNI. For OAI RAN, follow the strategy outlined at the specified link [30]. Download the OAI project and custom Docker files into the Docker directory, the following commands in Listing 7 suppose that both projects are in the home directory.

```
cp 5GDeployment/docker/Dockerfile.base.ubuntu22
    ↪ openairinterface5g/docker
cp 5GDeployment/docker/Dockerfile.build.ubuntu22.sim
    ↪  openairinterface5g/docker
cp 5GDeployment/docker/Dockerfile.gNB.ubuntu22.sim
    ↪ openairinterface5g/docker
cd openairinterface5g
docker build --target ran-base --tag ran-base:latest
    ↪  --file docker/Dockerfile.base.ubuntu22 .
docker build --target ran-build-sim --tag ran-build-
    ↪ sim:latest --file docker/Dockerfile.build.
    ↪ ubuntu22.sim .
docker build --target oai-gnb-sim --tag oai-gnb-sim:
    ↪ latest --file docker/Dockerfile.gNB.ubuntu22.
    ↪ sim .
docker tag oai-gnb-sim <dockerusername>/oai-gnb-sim
docker push <dockerusername>/oai-gnb-sim
cd 5gdeployment/charts/ran
helm install gnb oai-gnb --namespace oai-ran
```

Listing 7: Deployment of Open Air Interface RAN and core network

Verify the connection between the gNB and the Near-Real-Time RIC by checking the node states, as shown in Listing 8.

```
curl -s -X GET http://<ipe2mgr_service>:3800/v1/
    ↪ nodeb/states | jq .
```

Listing 8: Connection between RAN and Near RT-RIC

To test the gNB verify from the logs that it is correctly connected to the AMF function. If not connected it is possible that the gNB is not able to find the AMF inside the Kubernetes network. A quick workaround is to change the gNB `values.yaml` file by inserting the IP of the AMF service. This could be done by retrieving the Kubernetes cluster IP with `kubectl get pods -n oai-core` and by changing the `amfhost` value in the config parameter `amfhost: "your-cluster-ip"` To test that the UEs are correctly connected follow the guide available here [31] and execute the command in Listing 9 to check if they are correctly connected.

```
kubectl exec -it -n oai-ran -c nr-ue $(kubectl get
    ↪ pods -n oai-ran | grep oai-nr-ue | awk '{
    ↪ print $1}') -- ifconfig oaitun_ue1 | grep -E
    ↪ '(^|\s)inet($|\s)' | awk {'print $2'}
```

Listing 9: Test UEs connection

This setup ensures that the OAI gNB is properly integrated with the Near-RT RIC, facilitating efficient management and orchestration of the RAN components within the 5G network.

D. Non-RT RIC and SMO deployment

In this section, we provide two example deployments: one for the SMO and another for the Non-RT RIC. Both deployments are based on the K-release. The environment remains the same as in the previous tutorials. The first listing, 10, demonstrates the installation of the SMO, Non-RT RIC, and ONAP, while the second listing installs only the Non-RT RIC.

```
git clone --recursive "https://gerrit.o-ran-sc.org/r
    ↪ /it/dep"
./dep/smo-install/scripts/layer-0/0-setup-charts-
    ↪ museum.sh
./dep/smo-install/scripts/layer-0/0-setup-helm3.sh
./dep/smo-install/scripts/layer-1/1-build-all-charts
    ↪ .sh
./dep/smo-install/scripts/layer-2/2-install-oran.sh
kubectl get pods -n onap && kubectl get pods -n
    ↪ nonrtric && kubectl get pods -n smo
./smo-install/scripts/layer-2/2-install-simulators.
    ↪ sh

kubectl get pods --all-namespaces -o custom-columns=
    ↪ "NAME:.metadata.name"
```

Listing 10: SMO, Non-RT-RIC and ONAP deployment

In listing 10, there is an example deployment of the SMO that includes Non-RT RIC, ONAP, and SMO microservices. First, the repository is cloned to fetch the necessary files for installation (line 1). Next, ChartMuseum is set up to store Helm charts, and Helm 3 is installed to manage Kubernetes applications. All required Helm charts are built and uploaded to ChartMuseum for later use. The `2-install-oran.sh` script installs key components for the ORAN SMO system on Kubernetes. It first checks for a "flavour" argument to customize Helm configurations. The script then installs ONAP, Non-RT RIC, and SMO components using their respective Helm charts and custom configurations. After installation, it verifies the status of Kubernetes pods in the `onap`, `nonrtric`, and `smo` namespaces to ensure successful deployment. Finally, the O-RU O-DU and Topology server simulators are installed once all the pods are confirmed to be up and running.

[29]https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed/-/blob/master/docs/DEPLOY_SA5G_HC.md

[30]https://gitlab.eurecom.fr/oai/openairinterface5g/-/tree/develop/docker

[31]https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed/-/blob/master/docs/DEPLOY_SA5G_HC.md\#54-configure-oai-nr-ue-rfsimulator

```
git clone https://github.com/o-ran-sc/nonrtric.git
git checkout -b origin/k-release
docker compose -f docker-compose.yaml -f rapp/docker
    ↪ -compose.yaml -f a1-sim/docker-compose.yaml -
    ↪ f dmaap-mediator-go/docker-compose.yaml -f
    ↪ ics/docker-compose.yaml -f mr/docker-compose.
    ↪ yml -f sdnc/docker-compose.yml -f policy-
    ↪ service/docker-compose.yaml -f docker-compose
    ↪ -policy-framework/docker-compose.yml up -d
docker ps -a --format '{{.Names}}'
# Expected container names that should be up and
    ↪ running:
drools-apps, policy-distribution, drools, policy-
    ↪ xacml-pdp, policy-apex-pdp, policy-pap,
    ↪ policy-api, mariadb, policy-agent, a1-
    ↪ controller, sdnc-db, dmaap-mr, kafka,
    ↪ zookeeper, ics, consumer, dmaap-mediator-go,
    ↪ a1-sim-OSC, a1-sim-STD, a1-sim-STD-v2, r-app,
    ↪  wizardly_bohr
```

Listing 11: Non-RT RIC Deployment

Alternatively, it is possible to install only the Non-RT RIC using Docker Compose, as shown in Listing 11. This approach is simpler, more stable, and less prone to configuration errors compared to Kubernetes. However, it does not allow for the installation of the full SMO stack. The listing begins by cloning the Non-RT RIC repository and switching to the `k-release` branch. It then navigates to the `docker-compose` directory, where several Docker Compose files are used to deploy various services, including simulators, policy frameworks, and data services. The `docker compose` command is run with multiple configuration files to set up the services in detached mode. Finally, the `docker ps` command lists all running containers to verify the deployment, showing the names of the expected services.

Both deployments are connected to simulators for various interfaces, such as A1 and O1. They currently cannot connect to real-world implementations of Near-RT RIC, O-CU, O-DU, O-RU, and O-Cloud. This limitation arises from the incomplete integration of the projects and the complexity of managing the interfaces between these components.

E. Lessons Learned

This tutorial provides a step-by-step guide to setting up a complete RAN environment, focusing on deploying a Near-RT RIC, an example xApp, and a simulated and emulated gNB. Additionally, it covers deploying a SMO and Non-RT RIC blueprint with simulated A1 and O1 interfaces. The tutorial serves as an introduction to xApps and rApps development, testing, and evaluation, transitioning from a simulated setup to a real testbed. The hands-on experience includes setting up the O-RAN SC RIC as the Near-RT RIC, ns-O-RAN for simulation, and OAI for real testbed deployment. A key takeaway is the importance of selecting compatible tools, such as the O-RAN SC RIC for production environments and ns-O-RAN for full user-plane simulation. Moreover, it highlights the early stages of the Non-RT RIC and SMO projects, providing only an initial blueprint in Docker and Kubernetes, which does not allow connection to a simulated or real environment since all the interfaces connecting to the Near-RT RIC and gNB are simulated and simplified. The guide also emphasizes the need for a well-configured Kubernetes and Docker environment, proper software dependencies, and a structured approach to migrating xApps from simulation to real-world scenarios. The open challenges identified during this hands-on are detailed in Section VIII.

VIII. Research Gaps and Open Challenges

This section discusses the research and technical gaps currently present in the O-RAN environment and the ongoing challenges for the future of O-RAN. Interoperability and compliance with O-RAN specifications are discussed in Section VIII-A, while Section VIII-B presents the gaps in 5G-readiness of O-RAN software. The management and orchestration of O-RAN software and the separation of concerns between development and deployment are addressed in Sections VIII-C and VIII-D, respectively. For each theme addressed, we identify and present the discussion gaps observed in O-RAN software, including relevant aspects of simulators, emulators, real testbed implementations, and control plane software where applicable.

A. Interoperability and compliance of O-RAN specification

The interoperability and compliance of the discussed software with the O-RAN standards ensures compatibility across components. Ideally, if all the elements comply with the same standard, they could be switched around with minimal changes, enabling a plug-and-play environment.

The simulators analyzed have different levels of compliance with the O-RAN specifications. The two simulators with the highest compliance are 5G Toolbox [73] and ns-O-RAN [77]. 5G Toolbox implements a fully O-RAN-compliant version of its main functionality (O-RU and O-DU Packet CAPture (PCAP) trace generation), but it is noteworthy that only the RF and PHY layers can be simulated in this project. In contrast, ns-O-RAN [77] provides an O-RAN-compliant E2 interface and communicates using compliant messages, however, it does not implement the complete specification, only the KPM and RC service models with ASN.1 encoding. Overall, none of the simulators provides support for gNB disaggregation, O1 interface simulation, or integration, limiting the possibility of interoperability with real or simulated SMO/Non-RT RIC projects. The implementation of these features to comply with the O-RAN specification remains an open challenge within the state of the art.

The emulation and real testbed projects exhibit significant differences in their compliance with standards. At the PHY layer, the O-RAN SC project leverages Intel Flex RAN, ensuring compliance through that project, while srsRAN and OAI directly follow the 3GPP standard definitions, enabling integration with both commercial RUs and COTS UEs. Simu5G and FikoRE adhere to the channel model definitions, but their interactions with external software components are limited at this layer. At the MAC layer, full compliance is observed only in O-RAN SC, srsRAN, and OAI, whereas most software in the Simu5G and FikoRE tools does not meet this standard. The RLC layer is fully implemented by srsRAN and OAI, with other projects having partial, non-compliant implementations.

Support for different split options is crucial for interoperability, possibly allowing the integration of O-RUs, O-DUs, and O-CUs from various projects. This is particularly well-supported by srsRAN, which offers both split 7.2x and split 8.0, while OAI and O-RAN SC support only split 7.2x. The implementation of interfaces connecting to the RICs, specifically O1 and E2, is still incomplete across all projects. E2 is supported by srsRAN and OAI, while O1 can only be tested with O-RAN SC. Consequently, a collaboration between OAI and O-RAN SC has commenced to release a compliant O1 interface within OAI by the end of 2024. Another aspect that distinguishes projects like OAI, Simu5G, and srsRAN is their capacity to integrate and interact with different generations of devices, known as multi-Radio Access Technologies (RAT) support. In summary, while various projects exhibit different levels of readiness depending on the desired components, full compliance with both 3GPP standards and O-RAN Alliance specifications is still not fully achieved.

Concerning Near-RT RICs, all of them communicate with the RAN using the O-RAN-compliant E2 interface, and their compliance depends on the services they provide. Most Near-RT RICs provide a partial implementation of the provided services, as every project besides the embedded RIC implements at least the E2 termination, Shared Data Layer, messaging infrastructure, and API functions [25], [43], [44], [76]. The O-RAN SC RIC can be considered the most compliant project, providing all the services in the specification except the Y1 termination, and does not define custom service models [43]. The embedded RIC is a special case, as its compliance depends on the implementation of each xApp, and thus must be analyzed on a case-by-case basis [45]. In summary, no current project can be considered fully compliant with the O-RAN specification. However, we expect future versions of these Near-RT RICs to integrate all the services from the specification steadily.

Another critical challenge in achieving O-RAN compliance is the stability of the specification. As the O-RAN specification is relatively recent and currently in discussion, new versions of the specification are released often, and there are significant differences across versions. Hence, software projects compliant with a given version of the specification may not be compliant with the next one. For example, none of the Near-RT RIC projects has yet been able to implement a Y1 termination due to its recent inclusion in the standard. As some projects are not in continuous maintenance, they may not adapt to the new versions of the standard, and thus software considered O-RAN-compliant may not be interoperable with other O-RAN-compliant projects because they comply with different versions of the specification. While the O-RAN specification is expected to become more stable as it is more widely adopted and achieves higher technological maturity, it would also be desirable for O-RAN software projects to be aware of this phenomenon.

Thus, solutions for this issue, such as enabling O-RAN components to operate in *legacy* modes that refer to past versions of the standard to improve compatibility and interoperability, are a challenge that we expect state-of-the-art O-RAN software and literature to address in the future.

Regarding interoperability, two key gaps are the compatibility of xApps across different RICs and their software architecture. Currently, Near-RT RICs are implemented as individual platforms with their own SDKs that are not mutually compatible. Hence, xApps developed for a specific Near-RT RIC can only be executed in that platform, and using them in a different Near-RT RIC requires the xApp to be reimplemented in a different SDK. This phenomenon is also present across different versions of the same Near-RT RIC: although ColO-RAN RIC and O-RAN SC RIC are both based on different versions of the same Near-RT RIC, xApps for the O-RAN SC RIC need to be adapted to work under ColO-RAN RIC and vice versa [25], [43]. An approach to ease the transition across Near-RT RICs would be to have a common xApp software architecture, in which the xApp-RIC interaction, which is RIC-specific, is separated from the xApp's logic, which can be generic. Using such an architecture, developers could build their xApp using a RIC-agnostic approach, and porting an xApp across Near-RT RICs would only entail changing the components that govern the xApp-RIC interaction for those of the specific platform targeted. However, the O-RAN specification does not yet provide guidelines on the software architecture of xApps, and there is no *de facto* uniformity in their components or modules. We expect the O-RAN literature to tackle these interoperability challenges shortly.

Regarding interoperability between Non-RT RIC and SMO and their connectivity with the rest of the RAN, there is currently only one implementation of these two components by O-RAN SC. The development of the A1, O1, and O2 interfaces, which connect to the Near-RT RIC, gNB, and O-Cloud respectively, is ongoing. A significant effort by O-RAN SC involves dividing the implementation of these interfaces into separate projects for each endpoint, allowing external projects to adopt their implementation if they comply with O-RAN Alliance specifications. For instance, the A1 interface is implemented as two distinct components for both the Near-RT RIC and the Non-RT RIC, facilitating policy management. Similarly, the O1 interface is implemented within the O-DU project and for the SMO, while the O2 interface includes services for the SMO and the underlying infrastructure adhering to O-Cloud. Furthermore, O-RAN SC provides a simulator for each of the interfaces to facilitate the test phase of the SMO and Non-RT RIC However, external projects that aim to integrate with these network functions must adjust the communication of their components to align with the protocols adopted in those interfaces.

B. 5G-readiness

5G readiness in O-RAN consists of the capability of integrating existing solutions for simulation, emulation, and real testbed in real 5G deployment, eventually coexisting with previous generations. On the one hand, 5G-readiness at the lower layers of the RAN (e.g. RF, PHY, MAC) enables the performance measured at these layers to correctly mimic 5G performance, including its novelties such as mmWave technology. On the other hand, 5G readiness at higher layers (e.g. RRC, core) allows developers to leverage and test features specific to 5G, such as slicing. From the simulation and emulation

sides, 5G-readiness is desirable to allow performance testing of xApps, as well as to gather data, in environments as realistic as possible. Real testbed projects benefit from 5G-readiness by becoming off-the-shelf solutions for the implementation of 5G RANs.

In terms of simulation, integration with 5G in the upper layers is still an ongoing challenge. None of the six analyzed simulators provide support for complete 5G readiness. Neither gNB e2SM Emu or the SD-RAN simulator support the simulation of any 5G layer and are simply testing systems [75], [76], while the 5G Toolbox and POSS only simulate up to the PHY or MAC layers, respectively [73], [74]. The last two simulators, 5G-LENA [70] and ns-O-RAN [77], emulate the complete protocol stack, but from the RLC layer upwards, they use 4G protocols. Moreover, while they simulate a core, they only provide support for the 4G EPC. This limits the possibilities of testing 5G features such as slicing in simulated environments. Furthermore, as these two simulators cover the complete user plane, it is noteworthy that the integration is made with an emulated EPC, and not a concrete, real implementation of EPC, which may lead to some inconsistencies or required changes if the real EPC differs from its simulated counterpart. Nonetheless, 5G-LENA states that full 5G support is a planned feature that they expect to integrate in the future [70], and due to its implementation in ns-3, the simulated 5G layers may be possible to adapt for their use with ns-O-RAN as well.

From a real-testbed perspective, integration with the 5G core network has been successfully validated in the srsRAN [78] and OAI [79] projects, both of which are also capable of integrating with 4G EPC. This is particularly relevant for the O-CU-UP and O-CU-CP components, which require communication with the core via the the NG-U and NG-C interfaces. In contrast, emulation-focused solutions like FikoRE [90] and Simu5G [89] generally do not address this level of integration, although Simu5G can interface with 4G network components. Currently, the O-RAN SC does not support integration with any core network projects, whether 4G or 5G. However, collaboration has begun with OAI to incorporate the necessary components in future releases, with a particular focus on enabling the O-CU-UP and O-CU-CP components to communicate with the core via the NG-U and NG-C interfaces.

The integration of RAN simulators, emulators, and real testbeds with open-source core networks represents a significant challenge aligned with the 5G RAN openness principle. A preliminary step toward achieving this goal involves providing developers with documentation that details the configurations required to connect the RAN to the core network. While such documentation is available for OAI [79] and srsRAN [78], it is limited to a few projects. By enabling researchers and developers to test RAN projects with a broader range of open-source core projects, it is possible to obtain valuable insights into performance requirements, potentially leading to successful optimizations in communications by analyzing high-load scenarios. On the technical side, providing an interoperable interface that allows existing projects to connect to implementations of open-source 5G software, including but not limited to core networks, is crucial in providing 5G readiness. Alternatively, as in the case of simulators [70], projects can provide their own implementation, simulated or real, of the 5G functions and core. However, to ensure a seamless transition from simulated environments to emulated and real testbeds, it is necessary to ensure these implementations are interoperable or closely correspond, especially in terms of simulation.

C. Management and orchestration

The management and orchestration of elements within the O-RAN architecture are crucial for enabling O-RAN-based RANs to achieve greater flexibility, structure, and responsiveness to dynamic changes. Orchestration of O-RAN components facilitates their control, automated deployment, and scalability according to workload demands. Effective management also streamlines service deployment and can automate processes, allowing xApps and rApps to be deployed swiftly, either by human operators or automatically in response to specific events.

In the context of simulators, management and orchestration tasks focus on the ease of deployment, complexity, software isolation, and hardware requirements. Efficient management of simulators involves the ability to (de)-activate them on demand while ensuring logical isolation from other software, whether O-RAN-related or not, within the same environment. This can create challenges for projects integrating with O-RAN, particularly in maintaining up-to-date code. Given that these projects often rely on REST APIs, any changes or redefinitions in API behavior can lead to faults in applications developed by external users.

The simulators mostly support management and orchestration of isolated components: all simulators except POSS can be executed in containers [74], generally through Docker images, with gNB E2SM Emu, the SD-RAN simulator, and ns-O-RAN providing official support for this type of virtualization [75]–[77]. The SD-RAN simulator provides the highest support for management and orchestration by using the Helm package manager, as well as Kubernetes' native tools. This allows for the management and deployment of the SD-RAN simulator through simple commands, as well as to orchestrate it using Kubernetes' tools. Although custom support for Helm charts and Kubernetes is possible for other simulators, such as the custom Helm chart for ns-O-RAN used in Section VII, the widespread adoption of these technologies by O-RAN simulators is still a work in progress. Furthermore, it is noteworthy that the simulators provide interfaces for their management and orchestration through Kubernetes and the Helm package manager, but their automated orchestration, such as automated scaling or connection with other O-RAN software, is still an open challenge.

Emulators and real testbed projects exhibit varying stages of adoption for platforms managing and orchestrating applications. Emulators like Simu5G and FikoRE do not guarantee compatibility with platforms such as Kubernetes, Docker, or other open-source solutions. Their code has been tested only in bare-metal deployments, and Dockerfiles for container execution are not provided. Additionally, they do not

utilize a microservice architecture, rendering containerization efforts ineffective without a complete rethinking of the overall software architecture. In contrast, microservice architectures and corresponding Dockerfiles are available for O-RAN SC, srsRAN, and OAI. Consequently, O-RAN SC and srsRAN containers can be executed on Kubernetes, while OAI can be executed on OpenShift and Docker, with Docker Compose files also available. For managing complex deployments, srsRAN additionally leverages Helm as a package manager. Virtualization can also be complex. The kernel version is crucial when running 5G deployments, especially concerning the types of packages included. If emulator code is executed on virtual machines, there can be issues with virtualization modules not supporting these tools, a problem particularly encountered with OAI running on VM based on kvm.

Similarly, the support for orchestration and management from Near-RT RIC projects is still in early stages. While all Near-RT RICs are provided as one or more Docker images, enabling their execution in containers, most of these images are provided as-is, and their management and orchestration must be performed directly by their operators in their preferred container environment. The two Near-RT RIC projects that provide the highest management and orchestration support are the μONOS RIC [76] and the O-RAN SC RIC [43], both of which support the Helm package manager through charts and run in Kubernetes, although the μONOS RIC also provides a Makefile-based CLI for the management of its components. The key functionality provided by these two projects are Kubernetes operators, which integrate as part of Kubernetes logic to perform custom management of some elements. It is noteworthy that the O-RAN SC RIC's operator is an optional component and not yet part of the recommended setup. However, the main open challenge in the management and orchestration of Near-RT RICs is the scaling and verticalization of the platform. Although the operators are an important first step, the orchestrator, which is currently Kubernetes, remains unaware that it is being used to orchestrate Near-RT RIC services, as well as unaware of the requirements across services and between services and other O-RAN elements, which may lead to deployments with higher latencies or lower bandwidths than necessary. There is a need for vertical orchestrators, which may not necessarily be Kubernetes, that are aware of the entities running and can properly scale and orchestrate them.

The SMO and Non-RT RIC by O-RAN SC support deployment on Kubernetes using the Helm package manager, with Dockerfiles included for each project component. However, the installation guide remains fragmented, necessitating significant effort to achieve a stable deployment. Additionally, the project aims to foster multi-cloud support with ONAP, which is currently in its early stages.

The management and orchestration of the entire RAN infrastructure remain challenging due to inconsistencies in the principles and technologies adopted across various projects. The O-RAN Alliance seeks to address these challenges by defining the O-Cloud platform, which is closely tied to OpenStack, and through the efforts of its Cloudification and Orchestration Workgroup, WG06. However, this approach does not fully embrace one of the foundational principles of cloud engineering: the "as a Service" model, as well as a clear vision of which type of service architecture should be used: microservices or monolithic. The lack of comprehensive specifications has led to a fragmented landscape where projects independently decide whether to adopt containers and if so, whether to use Docker or Kubernetes, often without a clear, unified vision. This situation underscores the need for a more strategic approach to determine the appropriate "as a Service" model at each level of the RAN and to establish clear policies for system block orchestration. Although the SMO is intended to address these issues, its current implementation combines ONAP system components with existing RAN elements without a cohesive pervasive design. In response to these challenges, several papers propose enhancements to the standards, leveraging management and orchestration systems like OSM for 5G Core and RAN [98]–[100].

D. Separation of development and deployment concerns towards DevOps practices

Development and operations for extensive independent projects of O-RAN, such as simulators, emulators, real testbeds, and application-level components analyzed in this tutorial, can be challenging, especially when these components need to be integrated and executed together. The rapid pace of code implementation to align with specifications, along with frequent releases that necessitate extensive building and testing operations, can result in significant complexity and potential instability. Decoupling code development from operations related to building, packaging, documentation, deployment, and maintenance is essential and falls under DevOps practices. These practices are of paramount importance in the rapidly evolving landscape of open-source O-RAN projects, which often integrate code from various sources, as seen with xApps and rApps.

Ideally, an xApp or rApp developer should focus solely on the application logic and committing updates to a code repository, analogous to the development process for web or mobile applications [101]. Then, a Continuous Integration pipeline could automatically handle the deployment of O-RAN simulators or emulators, depending on the xApp o rApp target network function, that automatically test the validity of the code and evaluate its performance. Valid updates that exhibit an enhancement in RAN performance can then go through a Continuous Deployment pipeline to be automatically available in the real testbed. This allows a complete separation of the deployment concern, which can be handled by O-RAN operators by configuring simulators and emulators, the different pipelines code goes through, and the deployment of new xApps, rApps or updates to existing xApps and rApps in the real testbed, and the development concern, which is handled by developers that can focus exclusively on the business logic and coding of their software.

The currently available O-RAN simulators are working towards the goal of decoupling development and simulation, but their complete separation is still a challenge. Currently, the SD-RAN simulator [76] is leading in the adoption of

these practices, as its official distribution deploys the simulator, along with the μONOS RIC, with a single command, minimizing developers' concerns in simulator deployment. Nonetheless, there are two key shortcomings on the approach: first, the SD-RAN simulator only allows for code verification (i.e. integration testing regarding RIC and RAN communications), not performance evaluation on a RAN environment, and second, the SD-RAN simulator is coupled with the μONOS RIC, so developers who wish to use other Near-RT RICs cannot validate their xApps with this approach. The rest of the simulators either do not support xApp integration [70], [73] or are just distributed as-is, and the developer needs to manually configure the simulated scenarios, deploy the simulators, determine the IP address and ports used by the simulators' E2 interfaces, and deploy a compatible Near-RT RIC [74], [75], [77]. Some early works in the state of the art propose systems to make the process of simulator and Near-RT RIC deployment more automatic and straightforward to separate the concerns. An example of these proposals is our previous work named xSTART framework [102], which automates the deployment of ns-O-RAN and the ColO-RAN RIC together with an xApp under testing or evaluation. However, more comprehensive solutions that enable the deployment of different simulators and Near-RT RICs are still needed.

Emulation and real testbed projects are tackling DevOps tasks in diverse ways. All projects are hosted in Git repositories for version control, but only O-RAN SC manages software releases with comprehensive documentation and clear organization. Documentation of code dependencies, such as artifact registries and library states, remains incomplete. Build automation generally uses CMake, with OAI providing additional build scripts for the make command. None of the projects offer accessible CI/CD pipelines for end users, although O-RAN SC includes a `/ci` directory with Dockerfiles and build scripts. Unit tests are present in O-RAN SC, srsRAN, and OAI, but there is no automation for validating code or testing new releases alongside existing ones. Licensing is widely adopted across all analyzed projects. Documentation varies: OAI provides only deployment tutorials without theoretical explanations; srsRAN and FikoRE balance component explanations with tutorials, but often lack detail; O-RAN SC uses Confluence, a documentation website, and GitHub, though these can be inconsistent and outdated. Simu5G offers concise documentation with limited deployment tutorials for their emulator.

Efforts are underway to integrate various projects to enable the deployment of a complete RAN, as demonstrated in [13], [103]. Additionally, there are ongoing initiatives to define tools for specific DevOps phases, such as continuous testing [56]. However, these efforts often lack a comprehensive DevOps approach capable of maintaining up-to-date, verified, and tested versions of projects that are mutually compatible. While certain phases, such as *code planning*, are supported by tools like Confluence, and *coding* is facilitated by Git, there is a notable absence of tools to manage the *build phase*, such as Maven or Gradle. The *testing phase* in DevOps is often limited to unit tests, which test individual code blocks. A comprehensive testing approach should integrate functional tests, which deploy and test entire functionalities of the system, and system tests, which use simulated or emulated environments to test the behavior of the entire project set.

Tools for the *release phase* are already in an advanced stage, with some projects leveraging Jenkins. Existing projects partially address *deploy* and *operation phases* with Helm Operators. However, a comprehensive approach would also involve tools like Terraform, Vagrant, Ansible, Crossplane, Chef, and Puppet, which can integrate with various cloud providers and platforms like Docker and Kubernetes and provide a control layer for heterogeneous infrastructure through code.

The *monitoring phase* of DevOps is still not fully addressed by the projects. This phase involves controlling changes in the deployed DevOps pipelines. These pipelines should be monitored based on data collected from customer behavior and application performance to provide feedback on the overall system. This is generally achieved by adopting observability concepts, which include defining specific metrics for the application, logs related to RAN behavior, and traces of traffic exchanged between different RAN projects. Although this approach is gaining traction through the definition of monitoring interfaces like O1, it still lacks a comprehensive system-wide view. Moreover, we believe that, in this landscape, the concurrent adoption of tools for simulation, emulation, and real testbeds integrated into a clear cycle of development and operations can elevate the quality of the code proposed, enable reproducibility also within the research environment, and grant the release of secure, up-to-date, optimized solutions.

IX. Future Directions

This Section overviews current and expected future efforts within the O-RAN community to address gaps emerging from Section VIII and the associated open challenges.

Platform engineering for a Simulation, Emulation, Real Testbed continuum - O-RAN will necessitate a concerted integration effort to create a unified testbed that enables seamless transfer of features across environments, ensuring portability, smooth integration, and clear specifications throughout the process. Achieving coexistence and compatibility between simulated, emulated, and real testbeds by offering plug-and-play elements that can migrate between different environments will lower the entry barrier for new researchers and foster O-RAN adoption by telecom operators. We believe this can further evolve toward a Platform Engineering approach, which aims at creating a robust, scalable, reliable, and fully integrated RAN leveraging strictly defined DevOps pipelines for application, network, and infrastructure management.

Intent driven O-RAN - Managing diverse execution environments, along with numerous computational and networking services and resources, creates a highly complex scenario that is difficult to control programmatically and incurs significant costs. Therefore, automating this management is crucial. Intent-driven O-RAN represents a future development capable of addressing the management needs of various development environments, such as simulation, emulation, and real testbeds. It allows the specification of services and configurations at a high level while delegating their deployment and management during the execution cycle to the intent-driven system.

Furthermore, intent-driven can simplify the maintenance of Service Level Agreement (SLA) in line with business objectives and managing the services and resources used. Intent-driven systems will become crucial for abstracting, managing complexity, and fostering automation in the RAN.

AI-RAN with AIOps - O-RAN has not fully delivered the expected cost reductions, which has led many organizations, such as the AI-Alliance, to advocate for AI-RAN as a key enabler for cost reduction, particularly through Artificial Intelligence Operations (AIOps). Future directions for integrating AI and ML workflows into the RAN will focus on the AIOps pipelines, which should operate across three distinct control loop levels: i) Air Interface Control Loop for UEs Interaction: this loop ensures optimal connectivity and throughput by dynamically adapting to wireless conditions and user mobility; ii) RAN System Level Resource Management Control Loop (Orchestration): this loop optimizes resource allocation across the O-RU, O-DU, and O-CU architecture to ensure seamless user connectivity; iii) RAN Infrastructure and Operations Control Loop with Virtualization and AIOps: this loop guarantees network stability, performance, and resilience through intelligent automation. The role of AIOps, along with the architecture supporting these multi-level AI control loops, will continue to evolve to meet the demands of this complex scenario.

These future directions are expected to significantly enhance the openness of the RAN, promote RAN sharing among operators, and accelerate the transition toward an automated and AI-driven RAN.

X. Conclusion

The transition from closed-source, black-box RANs to Open RAN offers developers the ability to programmatically control networks through custom control plane applications. However, to effectively leverage this potential, developers must possess the knowledge to select the most appropriate evaluation environment, given the necessity to test and assess these applications in simulated, emulated, and real settings with a diversity of available RIC projects. Consequently, the transition from simulation, to emulation and consequently in the real testbed can be integrated into the DevOps process to enable a comprehensive testing environment for 5G. In this work, we present a comprehensive tutorial designed to guide developers through the O-RAN architecture, the various types of control applications they can create, and the range of open-source solutions for simulation, emulation, and real testbeds available for evaluation. We also explore the open-source RIC projects that can host these applications, detailing their architectures and compatibility with other O-RAN elements. The hands-on section of the tutorial enables developers to gain practical experience in setting up these evaluation environments using state-of-the-art projects, as well as transitioning their custom applications from simulated to emulated or real testbeds. Finally, we discuss current gaps in the literature on O-RAN application development, offering insights for future research within the O-RAN community.

The identified gaps in the current landscape include the need for 5G readiness, which involves integrating with other components of the 5G standard, such as core networks. Another challenge is the interoperability among various RAN projects that have been developed by different communities, as well as the management and orchestration of the entire 5G RAN infrastructure. Additionally, there is a need to better separate the development and deployment phases of these projects.

To address these issues, it is essential to implement 5G deployments that fully adhere to established standards. Providing comprehensive documentation, along with tutorials and compatibility tests, can also help ensure a more seamless integration process. Clear specifications that address both telecommunications and computing requirements for infrastructure and platforms are necessary to guide these efforts. Moreover, adopting concrete DevOps principles throughout the project lifecycle, including the concurrent use of simulation, emulation, and real testbed deployments, will improve the overall quality of the new software. This approach allows for more effective testing of new 5G capabilities and facilitates the incremental release of code updates, enhancing the overall lifecycle management of these projects and offering stable releases of the whole RAN to the end users.

Appendix: Abbreviations

3GPP	3rd Generation Partnership Project
4G	Fourth generation of mobile networks
5G	Fifth generation of mobile networks
5GC	5G Core
6G	Sixth generation of mobile networks
AI	Artificial Intelligence
AIOps	Artificial Intelligence Operations
API	Application Programming Interface
BBU	Baseband Unit
CD	Continuous Deployment
CI	Continuous Integration
CLI	Command Line Interface
CN	Core Network
COTS	Commercial Off The Shelf
CQI	Channel Quality Indicator
CU	Centralized Unit
dApp	Operator-defined Real Time Application
DevOps	Development and Operations
DRB	Data Radio Bearer
DRL	Deep Reinforcement Learning
DT	Digital Twin
DU	Distributed Unit
E2SM	E2 Service Model
E2SM-CCC	E2 Service Model for Cell Configuration and Control
E2SM-KPM	E2 Service Model for Key Performance Measurements
E2SM-NI	E2 Service Model for Network Interfaces
E2SM-RC	E2 Service Model for RAN Control

eNB	4G Evolved Node B
EPC	Evolved Packet Core
FAPI	5G Femto Application Platform Interface
FCAPS	Fault, configuration, accounting, performance and security
FFT	fast Fourier Transform
gNB	5G Next Generation Node B
iApp	Internal Application for Controller Specialization
KPI	Key Performance Indicator
KPM	Key Performance Measurement
LTE	Long-Term Evolution
MAC	Medium Access Control
MHO	Mobile HandOver
MIMO	Multiple Input Multiple Output
ML	Machine Learning
MLOps	Machine Learning Operations
Near-RT	Near-Real Time
Near-RT RIC	Near Real-Time RAN Intelligent Controller
NF	Network Function
nFAPI	network Functional Application Platform Interface
NFV	Network Function Virtualization
Non-RT	Non-Real Time
Non-RT RIC	Non Real-Time RAN Intelligent Controller
NR	New Radio
O-Cloud	O-Cloud
O-CU	Open Centralized Unit
O-CU-CP	Open Central Unit Control Plane
O-CU-UP	Open Central Unit User Plane
O-DU	Open Distributed Unit
O-RAN	Open Radio Access Network
O-RAN SC	O-RAN Software Community
O-RU	Open Radio Unit
O2 DMS	O2 Deployment Management Service
O2 IMS	O2 Infrastructure Management Service
OAI	Open Air Interface
OFDM	Orthogonal Frequency-Division Multiplexing
ONAP	Open Network Automation Platform
PCAP	Packet CAPture
PDCP	Packet Data Convergence Protocol
PHY	Physical
PHY-high	Physical-high
PHY-low	Physical-low
PM	Performance Management
QoS	Quality of Service
RAI	Radio Analytics Information
RAN	Radio Access Network
rApp	RAN Intelligent Controller Application
RAT	Radio Access Technologies
RC	RAN Control
RF	Radio Frequency
RIC	RAN Intelligent Controller
RLC	Radio Link Control
RMR	RIC Message Router
RRC	Radio Resource Control
RSRP	Reference Signal Received Power
RSRQ	Reference Signal Received Quality
RSSI	Received Signal Strength Indicator
RT	Real Time
RU	Radio Unit
SA	Stand Alone
SBA	Service Based Architecture
SCF	Small Cell Forum
SDAP	Service Data Adaptation Protocol
SDL	Shared Data Layer
SINR	Signal to Interference-plus-Noise Ratio
SLA	Service Level Agreement
SM	Service Model
SMO	Service Management and Orchestration
SRB	Signal Radio Bearer
srsRAN	Software Radio System RAN
UE	User Equipment
UHD	USRP Hardware Driver
USRP	Universal Software Radio Peripheral
VES	VNF Event Streaming
VESPAM	VNF Event Streaming Prometheus Adapter
VNF	Virtualized Network Function
xApp	Extended Application

ACKNOWLEDGMENTS

The European Union partially supported this work under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART") CUP: J33C22002880001.

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. [Online]. Available: https://doi.org/10.1109/COMST.2023.3239220

[2] O-RAN Alliance, "O-RAN: Towards an Open and Smart RAN," 2018. [Online]. Available: https://mediastorage.o-ran.org/white-papers/O-RAN.White-Paper-2018-10.pdf

[3] 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. [Online]. Available: https://doi.org/10.1016/j.comnet.2020.107516

[4] B. Tang, V. K. Shah, V. Marojevic, and J. H. Reed, "AI Testing Framework for Next-G O-RAN Networks: Requirements, Design, and Research Opportunities," *IEEE Wireless Communications*, vol. 30, no. 1, pp. 70–77, 2023. [Online]. Available: https://doi.org/10.1109/MWC.001.2200213

[5] M. Hoffmann, S. Janji, A. Samorzewski, L. Kułacz, C. Adamczyk, M. Dryjański, P. Kryszkiewicz, A. Kliks, and H. Bogucka, "Open RAN xApps Design and Evaluation: Lessons Learnt and Identified Challenges," *IEEE Journal on Selected Areas in Communications*, vol. 42, no. 2, pp. 473–486, 2024. [Online]. Available: https://doi.org/10.1109/JSAC.2023.3336190

[6] M. Agiwal, A. Roy, and N. Saxena, "Next Generation 5G Wireless Networks: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 1617–1655, 2016. [Online]. Available: https://doi.org/10.1109/COMST.2016.2532458

[7] M. Peng, Y. Sun, X. Li, Z. Mao, and C. Wang, "Recent Advances in Cloud Radio Access Networks: System Architectures, Key Techniques, and Open Issues," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2282–2308, 2016. [Online]. Available: https://doi.org/10.1109/COMST.2016.2548658

[8] M. F. Hossain, A. U. Mahin, T. Debnath, F. B. Mosharrof, and K. Z. Islam, "Recent research in cloud radio access network (C-RAN) for 5G cellular systems - A survey," *Journal of Network and Computer Applications*, vol. 139, pp. 31–48, 2019. [Online]. Available: https://doi.org/10.1016/j.jnca.2019.04.019

[9] P. K. Gkonis, P. T. Trakadas, and D. I. Kaklamani, "A Comprehensive Study on Simulation Techniques for 5G Networks: State of the Art Results, Analysis, and Future Challenges," *Electronics*, vol. 9, no. 3, 2020. [Online]. Available: https://doi.org/10.3390/electronics9030468

[10] V. S. Pana, O. P. Babalola, and V. Balyan, "5G radio access networks: A survey," *Array*, vol. 14, p. 100170, 2022. [Online]. Available: https://doi.org/10.1016/j.array.2022.100170

[11] A. Arnaz, J. Lipman, M. Abolhasan, and M. Hiltunen, "Toward Integrating Intelligence and Programmability in Open Radio Access Networks: A Comprehensive Survey," *IEEE Access*, vol. 10, pp. 67 747–67 770, 2022. [Online]. Available: https://doi.org/10.1109/ACCESS.2022.3183989

[12] L.-H. Shen, K.-T. Feng, and L. Hanzo, "Five Facets of 6G: Research Challenges and Opportunities," *ACM Comput. Surv.*, vol. 55, no. 11, feb 2023. [Online]. Available: https://doi.org/10.1145/3571072

[13] P. S. Upadhyaya, N. Tripathi, J. Gaeddert, and J. H. Reed, "Open AI Cellular (OAIC): An Open Source 5G O-RAN Testbed for Design and Testing of AI-Based RAN Management Algorithms," *IEEE Network*, vol. 37, no. 5, pp. 7–15, 2023. [Online]. Available: https://doi.org/10.1109/MNET.2023.3320933

[14] N. Aryal, E. Bertin, and N. Crespi, "Open Radio Access Network challenges for Next Generation Mobile Network," in *2023 26th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, 2023, pp. 90–94. [Online]. Available: https://doi.org/10.1109/ICIN56760.2023.10073507

[15] M. Amini, R. Stanica, and C. Rosenberg, "Where Are the (Cellular) Data?" *ACM Comput. Surv.*, vol. 56, no. 2, sep 2023. [Online]. Available: https://doi.org/10.1145/3610402

[16] W. Azariah, F. A. Bimo, C.-W. Lin, R.-G. Cheng, N. Nikaein, and R. Jana, "A Survey on Open Radio Access Networks: Challenges, Research Directions, and Open Source Approaches," *Sensors*, vol. 24, no. 3, 2024. [Online]. Available: https://doi.org/10.3390/s24031038

[17] J. Chen, X. Liang, J. Xue, Y. Sun, H. Zhou, and X. Shen, "Evolution of RAN Architectures Towards 6G: Motivation, Development, and Enabling Technologies," *IEEE Communications Surveys & Tutorials*, pp. 1–1, 2024. [Online]. Available: https://doi.org/10.1109/COMST.2024.3388511

[18] M. Polese, M. Dohler, F. Dressler, M. Erol-Kantarci, R. Jana, R. Knopp, and T. Melodia, "Empowering the 6G Cellular Architecture With Open RAN," *IEEE Journal on Selected Areas in Communications*, vol. 42, no. 2, pp. 245–262, Feb 2024. [Online]. Available: https://doi.org/10.1109/JSAC.2023.3334610

[19] S. Marinova and A. Leon-Garcia, "Intelligent O-RAN Beyond 5G: Architecture, Use Cases, Challenges, and Opportunities," *IEEE Access*, vol. 12, pp. 27 088–27 114, 2024. [Online]. Available: https://doi.org/10.1109/ACCESS.2024.3367289

[20] M. V. Ngo, N.-B.-L. Tran, H.-M. Yoo, Y.-H. Pua, T.-L. Le, X.-L. Liang, B. Chen, E.-K. Hong, and T. Q. Quek, "RAN Intelligent Controller (RIC): From open-source implementation to real-world validation," *ICT Express*, 2024. [Online]. Available: https://doi.org/10.1016/j.icte.2024.02.001

[21] M. Silva, J. P. Fonseca, D. P. Abreu, P. Martins, P. Duarte, R. Barbosa, B. Mendes, J. Silva, A. Goes, M. Araújo, B. Sousa, M. Curado, and J. Santos, "O-RAN and RIC Compliant Solutions for Next Generation Networks," in *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2023, pp. 1–7. [Online]. Available: https://doi.org/10.1109/INFOCOMWKSHPS57453.2023.10225994

[22] J. F. Santos, A. Huff, D. Campos, K. V. Cardoso, C. B. Both, and L. A. DaSilva, "Managing O-RAN Networks: xApp Development from Zero to Hero," 2024. [Online]. Available: https://arxiv.org/abs/2407.09619

[23] L. Bonati, S. D'Oro, M. Polese, S. Basagni, and T. Melodia, "Intelligence and Learning in O-RAN for Data-Driven NextG Cellular Networks," *IEEE Communications Magazine*, vol. 59, no. 10, pp. 21–27, 2021. [Online]. Available: https://doi.org/10.1109/MCOM.101.2001120

[24] A. S. Abdalla, P. S. Upadhyaya, V. K. Shah, and V. Marojevic, "Toward Next Generation Open Radio Access Networks: What O-RAN Can and Cannot Do!" *IEEE Network*, vol. 36, no. 6, pp. 206–213, 2022. [Online]. Available: https://doi.org/10.1109/MNET.108.2100659

[25] 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*, 2022. [Online]. Available: https://doi.org/10.1109/TMC.2022.3188013

[26] M. Polese, L. Bonati, S. D'Oro, P. Johari, D. Villa, S. Velumani, R. Gangula, M. Tsampazi, C. Paul Robinson, G. Gemmi, A. Lacava, S. Maxenti, H. Cheng, and T. Melodia, "Colosseum: The open ran digital twin," *IEEE Open Journal of the Communications Society*, vol. 5, pp. 5452–5466, 2024.

[27] A. Garcia-Saavedra and X. Costa-Pérez, "O-RAN: Disrupting the Virtualized RAN Ecosystem," *IEEE Communications Standards Magazine*, vol. 5, no. 4, pp. 96–103, 2021. [Online]. Available: https://doi.org/10.1109/MCOMSTD.101.2000014

[28] B. Brik, K. Boutiba, and A. Ksentini, "Deep Learning for B5G Open Radio Access Network: Evolution, Survey, Case Studies, and Challenges," *IEEE Open Journal of the Communications Society*, vol. 3, pp. 228–250, 2022. [Online]. Available: https://doi.org/10.1109/OJCOMS.2022.3146618

[29] V. Ranjbar, A. Girycki, M. A. Rahman, S. Pollin, M. Moonen, and E. Vinogradov, "Cell-free mmimo support in the o-ran architecture: A phy layer perspective for 5g and beyond networks," *IEEE Communications Standards Magazine*, vol. 6, no. 1, pp. 28–34, 2022. [Online]. Available: https://doi.org/10.1109/MCOMSTD.0001.2100067

[30] S. Lagén, L. Giupponi, A. Hansson, and X. Gelabert, "Modulation Compression in Next Generation RAN: Air Interface and Fronthaul Trade-offs," *IEEE Communications Magazine*, vol. 59, no. 1, pp. 89–95, 2021. [Online]. Available: https://doi.org/10.1109/MCOM.001.2000453

[31] A. Masaracchia, V. Sharma, M. Fahim, O. A. Dobre, and T. Q. Duong, "Digital Twin for Open RAN: Toward Intelligent and Resilient 6G Radio Access Networks," *IEEE Communications Magazine*, vol. 61, no. 11, pp. 112–118, 2023. [Online]. Available: https://doi.org/10.1109/MCOM.003.2200879

[32] V.-D. Nguyen, T. X. Vu, N. T. Nguyen, D. C. Nguyen, M. Juntti, N. C. Luong, D. T. Hoang, D. N. Nguyen, and S. Chatzinotas, "Network-Aided Intelligent Traffic Steering in 6G O-RAN: A Multi-Layer Optimization Framework," *IEEE Journal on Selected Areas in Communications*, vol. 42, no. 2, pp. 389–405, 2024. [Online]. Available: https://doi.org/10.1109/JSAC.2023.3336183

[33] S. Marinova and A. Leon-Garcia, "Intelligent O-RAN Beyond 5G: Architecture, Use Cases, Challenges, and Opportunities," *IEEE Access*, vol. 12, pp. 27 088–27 114, 2024. [Online]. Available: https://doi.org/10.1109/ACCESS.2024.3367289

[34] U. M. M, G. Sadashivappa, and R. Palepu, "Integration of RIC and xApps for Open-Radio Access Network for Performance Optimization," in *2023 7th International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS)*, 2023, pp. 1–4. [Online]. Available: https://doi.org/10.1109/CSITSS60515.2023.10334159

[35] T. Hai, Y. Ning, Z. Zhi, D. Zhongda, and S. Jia, *5G NR and Enhancements: From R15 to R16*. Elsevier, 2022. [Online]. Available: https://doi.org/10.1016/C2020-0-04150-2

[36] 3GPP, Sophia Antipolis, France, "NR and NG-RAN Overall description;," 3rd Generation Partnership Project (3GPP), Technical report (TR) 38.300, 2017. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3191

[37] ——, "Study on new radio access technology: Radio access architecture and interfaces," 3rd Generation Partnership Project (3GPP), Technical report (TR) 38.801, 2024. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3056

[38] L. M. P. Larsen, A. Checko, and H. L. Christiansen, "A Survey of the Functional Splits Proposed for 5G Mobile Crosshaul Networks," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 146–172, 2019. [Online]. Available: https://doi.org/10.1109/COMST.2018.2868805

[39] SCF, "Small Cell Forum (SCF)," Small Cell Forum (SCF), Technical specification (TS), 2010. [Online]. Available: https://www.smallcellforum.org/

[40] 3GPP, Sophia Antipolis, France, "System architecture for the 5G System (5GS)," 3rd Generation Partnership Project (3GPP), Technical specification (TS) 23.501, 2023. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144

[41] A. Lacava, L. Bonati, N. Mohamadi, R. Gangula, F. Kaltenberger, P. Johari, S. D'Oro, F. Cuomo, M. Polese, and T. Melodia, "dApps: Enabling Real-Time AI-based Open RAN control," *Computer Networks*, p. 111342, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128625003093

[42] Open Network Operating System (ONOS). (visited on Aug. 28, 2023). [Online]. Available: https://opennetworking.org/onos/

[43] O-RAN Alliance, "O-RAN Software Community," 2024. [Online]. Available: https://o-ran-sc.org/

[44] R. Schmidt, M. Irazabal, and N. Nikaein, "FlexRIC: an SDK for next-generation SD-RANs," in *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, 2021, pp. 411–425.

[45] Open AI Cellular Consortium, "Open AI Cellular," 2022. [Online]. Available: https://openaicellular.github.io/oaic/

[46] Atlassian, "Jira software," https://confluence.atlassian.com/jira061, 2024, accessed: 2024-10-15.

[47] ——, "Confluence," https://support.atlassian.com/confluence-cloud/, 2024, accessed: 2024-10-15.

[48] S. Chacon and B. Straub, "Git," https://git-scm.com/, 2024, accessed: 2024-10-15.

[49] Jenkins Project, "Jenkins," https://www.jenkins.io/doc/, 2024, accessed: 2024-10-15.

[50] The Linux Foundation, "O-RAN Software Community," 2024. [Online]. Available: https://wiki.o-ran-sc.org/

[51] CMake Developers, "Cmake documentation," https://cmake.org/documentation/, 2024, accessed: 2024-10.

[52] Gradle, Inc., "Gradle build tool," https://gradle.org/, 2024, accessed: 2024-10-15.

[53] Apache Software Foundation, "Apache maven," https://maven.apache.org/, 2024, accessed: 2024-10-15.

[54] Google, "Google test," https://github.com/google/googletest, 2024, accessed: 2024-10-15.

[55] JUnit Team, "Junit," https://junit.org/junit5/, 2024, accessed: 2024-10-15.

[56] L. Bonati, M. Polese, S. D'Oro, P. B. del Prever, and T. Melodia, "5G-CT: Automated Deployment and Over-the-Air Testing of End-to-End Open Radio Access Networks," *IEEE Communications Magazine*, vol. 63, no. 1, pp. 155–160, 2025. [Online]. Available: https://doi.org/10.1109/MCOM.001.2300675

[57] GitLab Inc., "Gitlab ci/cd," https://docs.gitlab.com/ee/ci/, 2024, accessed: 2024-10-15.

[58] T. K. Authors, "Kubernetes: Production-grade container orchestration," 2023, accessed: 2025-02-21. [Online]. Available: https://kubernetes.io/

[59] M. Suman, R. Patil, and R. Kotoju, "Unit Test Framework for Distributed Unit Manager (DUMGR) of 5G RAN," in *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, 2023, pp. 1–6. [Online]. Available: https://doi.org/10.1109/ICCCNT56998.2023.10306895

[60] O. Arouk and N. Nikaein, "Kube5G: A Cloud-Native 5G Service Platform," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–6. [Online]. Available: https://doi.org/10.1109/GLOBECOM42002.2020.9348073

[61] M. Dryjański, L. Kulacz, and A. Kliks, "Toward Modular and Flexible Open RAN Implementations in 6G Networks: Traffic Steering Use Case and O-RAN xApps," *Sensors*, vol. 21, no. 24, 2021. [Online]. Available: https://doi.org/10.3390/s21248173

[62] P. Authors, "Prometheus: Monitoring system & time series database," 2025, accessed: 2025-02-21. [Online]. Available: https://prometheus.io

[63] G. Labs, "Grafana: The open observability platform," 2025, accessed: 2025-02-21. [Online]. Available: https://grafana.com

[64] X. Vasilakos, B. Köksal, D. H. Izaldi, N. Nikaein, R. Schmidt, N. Ferdosian, R. F. Sari, and R.-G. Cheng, "ElasticSDK: A Monitoring Software Development Kit for enabling Data-driven Management and Control in 5G," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–7. [Online]. Available: https://doi.org/10.1109/NOMS47738.2020.9110373

[65] S. Barrachina-Muñoz, M. Payaró, and J. Mangues-Bafalluy, "Cloud-native 5G experimental platform with over-the-air transmissions and end-to-end monitoring," in *2022 13th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*, 2022, pp. 692–697. [Online]. Available: https://doi.org/10.1109/CSNDSP54353.2022.9908028

[66] AI-RAN Alliance, "AI-RAN Alliance: Vision and Mission White Paper ," AI-RAN Alliance, Technical Report (TR) 1.00, 2024. [Online]. Available: https://ai-ran.org/wp-content/uploads/2024/12/AI-RAN_Alliance_Whitepaper.pdf

[67] Arnaz, Azadeh and Lipman, Justin and Abolhasan, Mehran and Hiltunen, Matti, "Toward Integrating Intelligence and Programmability in Open Radio Access Networks: A Comprehensive Survey," *IEEE Access*, vol. 10, pp. 67 747–67 770, 2022.

[68] Peizheng Li and Ioannis Mavromatis and Tim Farnham and Adnan Aijaz and Aftab Khan, "Adapting MLOps for Diverse In-Network Intelligence in 6G Era: Challenges and Solutions," 2024. [Online]. Available: https://arxiv.org/abs/2410.18793

[69] H.-T. Thieu, V.-Q. Pham, A. Kak, and N. Choi, "Demystifying the Near-real Time RIC: Architecture, Operations, and Benchmarking Insights," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2023, pp. 1–8. [Online]. Available: https://doi.org/10.1109/INFOCOMWKSHPS57453.2023.10225852

[70] 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

[71] M. Mezzavilla, M. Zhang, M. Polese, R. Ford, S. Dutta, S. Rangan, and M. Zorzi, "End-to-end simulation of 5G mmWave networks," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2237–2263, 2018. [Online]. Available: https://doi.org/10.1109/COMST.2018.2828880

[72] J. Lee, S. Jung, S.-E. Hong, and H. Lee, "Development on open-ran simulator with 5g-lena," in *2024 International Conference on Information Networking (ICOIN)*. IEEE, 2024, pp. 176–178.

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

[74] T. Karamplias, S. T. Spantideas, A. E. Giannopoulos, P. Gkonis, N. Kapsalis, and P. Trakadas, "Towards closed-loop automation in 5g open ran: Coupling an open-source simulator with xapps," in *2022 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. IEEE, 2022, pp. 232–237. [Online]. Available: https://doi.org/10.1109/EuCNC/6GSummit54941.2022.9815658

[75] E. Moro, "gNB-e2sm-emu: gNB emulator for custom E2 SM testing," 2023. [Online]. Available: https://github.com/ANTLab-polimi/gNB-e2sm-emu

[76] O. Sunay, S. Ansari, S. Condon, J. Halterman, W. Kim, R. Milkey, G. Parulkar, L. Peterson, A. Rastegarnia, and T. Vachuska, "SD.RAN: ONF's Software-Defined RAN Platform Consistent with the O-RAN Architecture," Open Networking Foundation, Tech. Rep., August 2020. [Online]. Available: https://opennetworking.org/wp-content/uploads/2020/08/SD-RAN-v2.0.pdf

[77] A. Lacava, M. Bordin, M. Polese, R. Sivaraj, T. Zugno, F. Cuomo, and T. Melodia, "ns-O-RAN: Simulating O-RAN 5G Systems in ns-3," in *Proceedings of the 2023 Workshop on Ns-3*, ser. WNS3 '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 35–44. [Online]. Available: https://doi.org/10.1145/3592149.3592161

[78] Software Radio Systems, "srsRAN Project," 2024. [Online]. Available: https://docs.srsran.com/projects/project/en/latest/

[79] EURECOM, "Open Air Interface RAN," 2024. [Online]. Available: https://openairinterface.org/oai-5g-ran-project/

[80] G. Nardini, G. Stea, and A. Virdis, "Simu5G," 2024. [Online]. Available: https://simu5g.org/users-guide/emulation

[81] D. G. Morin, M. J. Lopez-Morales, P. Perez, A. G. Armada, and A. Villegas, "FikoRE," 2024. [Online]. Available: https://github.com/nokia/5g-network-emulator.git

[82] O-RAN Alliance, "O-RAN Control, User and Synchronization Plane Specification," O-RAN Alliance, Technical Specification (TS) 16.01, 2024. [Online]. Available: https://specifications.o-ran.org/download?id=738

[83] ——, "O-RAN Base Station O-DU and O-CU Software Architecture and APIs," O-RAN Alliance, Technical Specification (TS) 13.00, 2024. [Online]. Available: https://specifications.o-ran.org/download?id=760

[84] ——, "O-RAN Acceleration Abstraction Layer General Aspects and Principles," O-RAN Alliance, Technical Specification (TS) 10.00, 2024. [Online]. Available: https://specifications.o-ran.org/download?id=753

[85] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, "srsLTE: an open-source platform for LTE evolution and experimentation," in *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, ser. WiNTECH '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 25–32. [Online]. Available: https://doi.org/10.1145/2980159.2980163

[86] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "OpenAirInterface: A Flexible Platform for 5G Research," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, p. 33–38, oct 2014. [Online]. Available: https://doi.org/10.1145/2677046.2677053

[87] 3GPP, Sophia Antipolis, France, "NG-RAN; F1 Application Protocol (F1AP)," 3rd Generation Partnership Project (3GPP), Technical specification (TS) 38.473, 2024. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3260

[88] ——, "General Packet Radio System (GPRS) Tunnelling Protocol User Plane (GTPv1-U)," 3rd Generation Partnership Project (3GPP), Technical specification (TS) 29.281, 2024. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1699

[89] G. Nardini, D. Sabella, G. Stea, P. Thakkar, and A. Virdis, "Simu5G–An OMNeT++ Library for End-to-End Performance Evaluation of 5G Networks," *IEEE Access*, vol. 8, pp. 181 176–181 191, 2020. [Online]. Available: https://doi.org/10.1109/ACCESS.2020.3028550

[90] D. G. Morin, M. J. Lopez-Morales, P. Perez, A. G. Armada, and A. Villegas, "FikoRE: 5G and Beyond RAN Emulator for Application Level Experimentation and Prototyping," *IEEE Network*, vol. 37, no. 4, pp. 48–55, 2023. [Online]. Available: https://doi.org/10.1109/MNET.002.2200595

[91] 3GPP, Sophia Antipolis, France, "NR; Physical channels and modulation," 3rd Generation Partnership Project (3GPP), Technical specification (TS) 38.211, 2024. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3213

[92] ——, "NR; Physical layer procedures for data," 3rd Generation Partnership Project (3GPP), Technical specification (TS) 38.214, 2024. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3216

[93] ——, "NR; User Equipment (UE) radio access capabilities," 3rd Generation Partnership Project (3GPP), Technical specification (TS) 38.306, 2024. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3193

[94] ——, "Study on channel model for frequencies from 0.5 to 100 GHz," 3rd Generation Partnership Project (3GPP), Technical report (TR) 38.901, 2024. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3173

[95] ——, "Telecommunication management; Performance measurement: File format definition," 3rd Generation Partnership Project (3GPP), Technical specification (TS) 32.432, 2024. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2014

[96] ——, "Telecommunication management; Performance measurement; eXtensible Markup Language (XML) file format definition," 3rd Generation Partnership Project (3GPP), Technical specification (TS) 32.435, 2024. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2017

[97] A. Feraudo, S. Maxenti, A. Lacava, P. Bellavista, M. Polese, and T. Melodia, "xdevsm: Streamlining xapp development with a flexible framework for o-ran e2 service models," in *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, ser. ACM MobiCom '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1954–1961. [Online]. Available: https://doi.org/10.1145/3636534.3697325

[98] C. Parada, J. Bonnet, E. Fotopoulou, A. Zafeiropoulos, E. Kapassa, M. Touloupou, D. Kyriazis, R. Vilalta, R. Muñoz, R. Casellas, R. Martínez, and G. Xilouris, "5Gtango: A Beyond-Mano Service Platform," in *2018 European Conference on Networks and Communications (EuCNC)*, 2018, pp. 26–30. [Online]. Available: https://doi.org/10.1109/EuCNC.2018.8443232

[99] F. Malandrino, C. F. Chiasserini, C. Casetti, G. Landi, and M. Capitani, "An Optimization-Enhanced MANO for Energy-Efficient 5G Networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1756–1769, 2019. [Online]. Available: https://doi.org/10.1109/TNET.2019.2931038

[100] K. Katsalis, J. Triay, H. Zafar, Z. Yousaf, J. Pieczerak, L. Deng, and B. Chatras, "NFV-MANO Support for Orchestration and Management of the Virtualized RAN," in *2023 IEEE Conference on Standards for Communications and Networking (CSCN)*, 2023, pp. 339–345. [Online]. Available: https://doi.org/10.1109/CSCN60443.2023.10453188

[101] S. Laso, J. Berrocal, P. Fernández, A. Ruiz-Cortés, and J. M. Murillo, "Perses: A framework for the continuous evaluation of the QoS of distributed mobile applications," *Pervasive and Mobile Computing*, vol. 84, p. 101627, 2022. [Online]. Available: https://doi.org/10.1016/j.pmcj.2022.101627

[102] J. L. Herrera Gonzalez, S. Montebugnoli, D. Scotece, and L. Foschini, "xSTART: xApp Simulated Evaluation Environment for Developers," in *2024 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0 & IoT)*, 2024, pp. 366–371. [Online]. Available: https://doi.org/10.1109/MetroInd4.0IoT61288.2024.10584214

[103] S. Maxenti, R. Shirkhani, M. Elkael, L. Bonati, S. D'Oro, T. Melodia, and M. Polese, "AutoRAN: Automated and Zero-Touch Open RAN Systems," 2025. [Online]. Available: https://arxiv.org/abs/2504.11233

AUTHOR BIOGRAPHIES

Juan Luis Herrera received a Ph.D. in Computer Science from the University of Extremadura in 2023. He is a researcher in the University of Bologna's Computer Science and Engineering Department. His main research interests include IoT, fog computing, SDN, and O-RAN.

Sofia Montebugnoli (Student Member, IEEE) is a Ph.D. candidate in Computer Science and Engineering from the University of Bologna. She received a Master's Degree in Computer Engineering from the University of Bologna in 2022. Her main research interests include O-RAN, cloud continuum, and middleware.

Domenico Scotece (Member, IEEE) is a junior assistant professor at the University of Bologna, Italy, right after having obtained the Ph.D degree at the same University in April 2020. His research interests include pervasive computing, middleware for fog and edge computing, the Software-Defined Networking, the Internet of Things, and 5G network planning and design.

Luca Foschini (Senior Member) received a Ph.D. degree in computer science engineering from the University of Bologna, Italy, where he is now a full professor of distributed systems. His interests span from integrated management of distributed systems and services to mobile crowd-sourcing/-sensing, from infrastructures for Industry 4.0 to fog/edge cloud systems.

Paolo Bellavista (Senior Member, IEEE) received a Ph.D. in computer science engineering from the University of Bologna, Italy, where he is now a full professor of web technologies and mobile systems. His research activities span from pervasive wireless computing to edge cloud computing or middleware for Industry 4.0 applications.