Open Radio Access Networks (O-RAN) Experimentation Platform: Design and Datasets

J. Xavier Salvat, Jose A. Ayala-Romero, Lanfranco Zanzi, *Member, IEEE*,
Andres Garcia-Saavedra, and Xavier Costa-Perez, *Senior Member, IEEE*

***Abstract*—The Open Radio Access Network (O-RAN) Alliance is driving the latest evolution of RAN deployments, moving from traditionally closed and dedicated hardware implementations towards virtualized instances running over shared platforms characterized by open interfaces. Such progressive decoupling of radio software components from the hardware paves the road for future efficient and cost-effective RAN deployments. Nevertheless, there are many open aspects towards the successful implementation of O-RAN networks, such as the real-time configuration of the network parameters to maximize performance, how to reliably share processing units among multiple virtualized base station (vBS) instances, how to palliate their energy consumption, or how to deal with the couplings between vRANs and other services co-located at the edge. Intending to shed light on these aspects, in this article, we showcase the design principles of an O-RAN compliant testbed, and present different datasets collected over a wide set of experiments, which are made public to foster research in this field.**

***Index Terms*—O-RAN, vRAN, RAN Intelligent Control.**

I. Introduction

The O-RAN Alliance is a joint effort in the mobile industry to redesign the future Radio Access Network (RAN) technologies [1]. The key principles are threefold: (i) intelligent RAN control at different timescales to foster innovation; (ii) open interfaces between control-plane components and network functions to break the traditional vendor lock-in; and (iii) virtualization to improve flexibility and reduce costs.

However, the advent of O-RAN raises novel technical challenges. First, the higher level of flexibility comes at the cost of less predictable performance and computing resource demand. In contrast to the traditional hardwired base stations (BSs), the computing resources needed by a virtualized BS (vBS) vary with the context, including network load, the modulation and coding scheme (MCS) used, channel quality, etc. The mapping between this high-dimensional set of parameters and the requirements for computing resources or energy consumption is very complex and hard to predict [2].

Second, virtualizing RAN functions over a shared infrastructure can provide high flexibility and cost efficiency, but the overhead introduced when contending for a shared resource compromises reliability to execute signal processing tasks within tight time deadlines. In fact, different works show that resource contention between NFs sharing computing infrastructure may lead to up to 40% of performance degradation compared to dedicated platforms [3] This coupling between radio resource allocation and computing requirements poses new challenges [2].

Third, RAN virtualization poses a new energy consumption profile compared to traditional BSs that operate with dedicated hardware [4]. The energy consumption of vBSs not only depends on the network state (e.g., traffic load, SNR), but also on the general-purpose hardware (e.g., CPU/GPU) and the software implementation of the radio stack.

Finally, when considering AI services running at the edge of the network, both the edge and network configuration are intertwined [5]. That is, the configuration of the edge services (e.g., QoS) and the network (e.g., channel capacity) jointly impact both service performance and power consumption of the whole system. Therefore, evaluating and orchestrating the system at once, although challenging, can bring global benefits in terms of performance and energy.

To shed light on these aspects, we present an O-RAN testbed that provides a prototypical environment to experiment with different network settings and evaluate machine learning (ML) solutions to the above problems. Using this testbed, we collect three datasets aimed at contributing to different relatively-unexplored aspects of O-RAN. The datasets are publicly available at [6] and are described as follow:

- *Computing dataset* characterizes the computing usage of vBS as a function of several contextual (e.g., traffic load, channel quality) and configuration (e.g., MCS, CPU time) parameters. We also evaluate the effect of several vBS instances sharing the same platform [2].
- *Energy dataset* measures the energy consumption of a vBS as a function of a wide range of parameters (e.g., MCS, airtime, computing platform, bandwidth). The energy measurements are taken in parallel using software tools and an external digital power meter [4], [7].
- *Application dataset* considers an AI service running in an edge server. It characterizes at the same time the service performance and the consumed energy of the vBS and edge server as a function of their joint configuration [5].

These datasets are the result of the study of different problems addressed in our previous work. In particular, [2] proposes a deep reinforcement learning approach to allocate the computing resources of a virtualized RAN (vRAN). In [7], the energy consumption of the uplink is studied and characterized using an analytical model. In [4], a Bayesian learning

J. X. Salvat, J. A. Ayala, L. Zanzi, and A. Garcia-Saavedra are with NEC Laboratories Europe GmbH, Heidelberg, Germany.

X. Costa-Pérez is with NEC Laboratories Europe GmbH, Heidelberg, Germany, and i2CAT Foundation and ICREA, Barcelona, Spain.

The work was supported by the European Commission through Grants No. SNS-JU-101097083 (BeGREEN) and 101017109 (DAEMON). Additionally, it has been supported by MINECO/NG EU (No. TSI-063000-2021-7) and the CERCA Programme.

algorithm is proposed to allocate vRAN radio resources, balancing energy and performance. Finally, [5] uses online learning to jointly configure the vRAN and an edge AI service to save energy while providing performance guarantees.

Other related works consider the deployment of vRANs on commodity hardware [8], [9]. The authors in [8] propose a CPU scheduling framework to collocate the vRAN with general-purpose workloads while meeting the latency requirements. In [9], an optimized data processing pipeline is proposed to handle the high computational demand of massive MIMO processing in software-only systems. Finally, ColO-RAN [10] presents an SDR-enabled large-scale framework to test O-RAN RIC algorithms. For example, OrchestRAN [11], a RIC algorithm that orchestrates other data-driven algorithms based on mobile operators' intents, is prototyped and evaluated in ColO-RAN.

II. O-RAN Architecture

Fig. 1 provides an overview of the O-RAN architecture. Like 3GPP, O-RAN distributes all the functions of a gNB across three main Network Functions (NFs): (i) a Radio Unit (O-RU), (ii) a Distributed Unit (O-DU), and (iii) a Central Unit (O-CU) [12]. The O-RU hosts the lowest physical layer (PHY) tasks, including amplification, signal sampling, and FFT operations; the O-DU hosts the RLC, MAC, and higher PHY operations such as forward error correction (FEC). Finally, the O-CU accommodates the RRC, SDAP, and PDCP layers. In addition, O-RAN specifies an O-Cloud platform to host virtualized NFs (VNFs), including an acceleration abstraction layer (AAL) to offload signal processing operations such as FEC or FFT.

In the control plane, O-RAN introduces a non-real-time RAN intelligent controller (non-RT RIC), and a near-real-time RAN intelligent controller (near-RT RIC). The non-RT RIC is hosted by the Service Management and Orchestration (SMO) framework and enables control loops at large time scales (i.e., seconds or minutes). Formally, the different control applications that run within the non-RT RIC are called rApps, and they support different tasks such as analyzing RAN monitoring information or issuing control policies. Conversely, the near-RT RIC supports control loops over sub-second time scales (i.e., $\sim$10ms) through the so-called xApps.

O-RAN defines four key interfaces –O1, A1, E2, and O2– which allow information exchange among the components of the architecture. Specifically, the O1 interface enables operation and management procedures, such as FCAPS (Fault, Configuration, Accounting, Performance, and Security), and software and file management. The A1 interface connects the non-RT RIC with the near-RT RIC and enables the enforcement of control policies defined at the upper architectural levels. The near real-time RIC connects to the O-gNB components (O-CU, O-DU, and O-RU) by means of the E2 interface, enabling the enforcement of control policies and data collection. Finally, the O2 interface connects the SMO with the O-Cloud to enable infrastructure monitoring and management. On the other hand, O-RAN leverages on the 3GPP fronthaul interfaces, which are an enabler of the gNB disaggregated architecture.

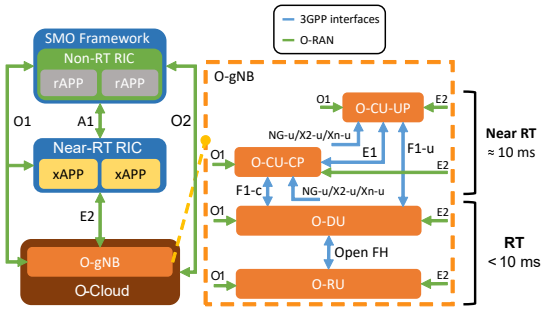

Fig. 1. O-RAN architecture. On the left, the O-gNB and the O-Cloud are shown with the O-RAN RAN Intelligent Controllers (RICs). On the right, a detailed scheme of the O-gNB functionalities.

III. Testbed Design and Implementation

This section presents an O-RAN compliant testbed that enables experimentation with vRAN deployments and evaluation of resource allocation and orchestration algorithms. We also detail its design principles and its implementation. Fig. 2 depicts the main functional blocks and overall architecture, while Fig. 3 shows the real testbed.

A. Virtualized RAN Computing Platform

As depicted in Fig. 2, the testbed hosts (1) multiple user equipments (UEs), each one attached to (2) a virtualized vBS instance running in (3) a shared computing platform. Each UE consists of a radio head and a set of dedicated computing resources provided by a laptop. Such resources host the complete radio protocol stack and processes from heterogeneous mobile applications. Both UEs and vBSs use a USRP B210 board as a radio head, and the srsRAN [13] software to implement the radio protocol stack. The vBSs' USRP boards are attached to the computing pool via a USB3.0 connector, while the srsRAN vBSs run as containerized software instances using `Docker`. To ensure repeatability, UEs and vBSs' radio front-ends are connected with RF SMA cables and 20dB attenuators. Each UE is connected to one vBS, emulating the aggregated traffic volumes generated over a cell. In our testbed, we support a maximum of 5 UEs and 5 vBS.

The computing platform (3) features commercial off-the-shelf components like an 8 cores Intel i7-7700K CPU and Ubuntu operating system (OS). For the purposes of our tests, the kernel has been compiled with the option of `CONFIG_RT_GROUP_SCHED`, so that resource allocation can be performed on real-time threads. 6 computing cores are reserved for the shared pool by means of `systemd`'s `CPUAffinity`. Specifically, considering the CPU's topology depicted in Fig. 4, we use cores 0 and 4 to run the OS' processes and cores 1–3 and 5–7 to run the Docker instances containing vBSs. Thus, the access to the L1–L2 caches of both core sets is isolated, minimizing the residual computing noise coming from the OS.

B. Service Management and Orchestrator and Mobile Core

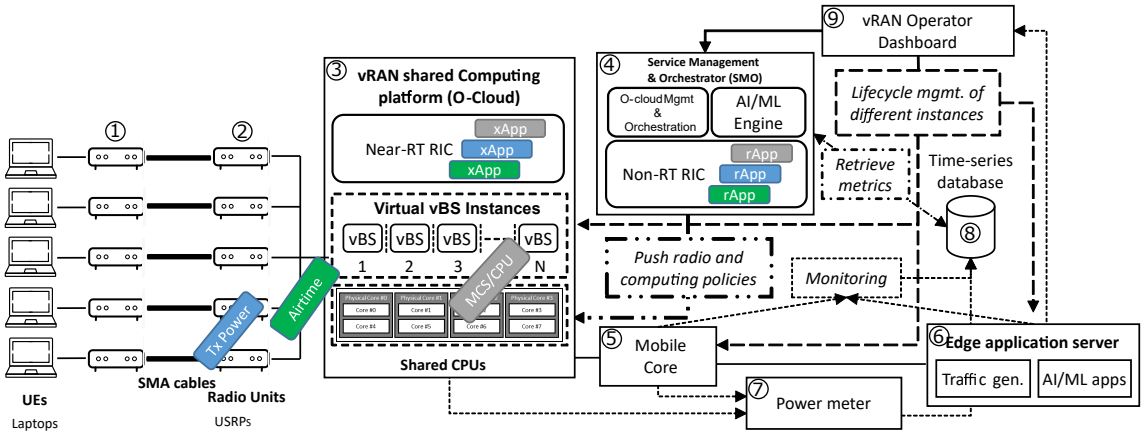

Fig. 2. Detailed testbed architecture.

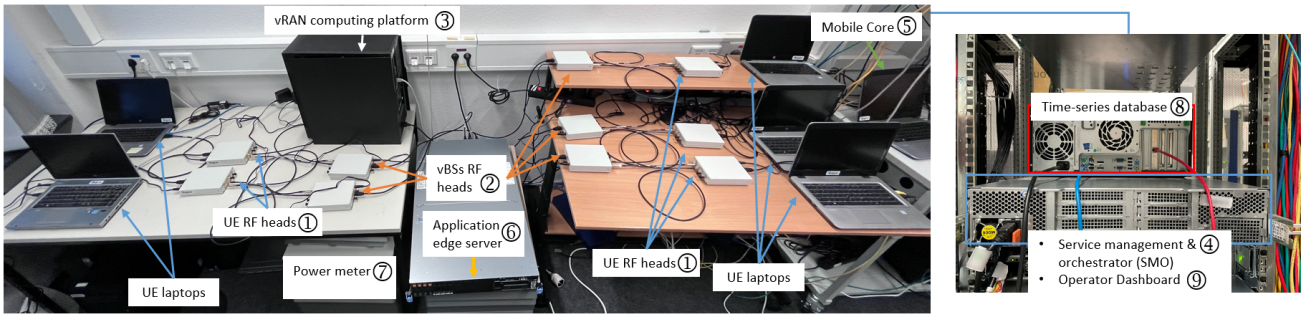

Fig. 3. Picture of the testbed.

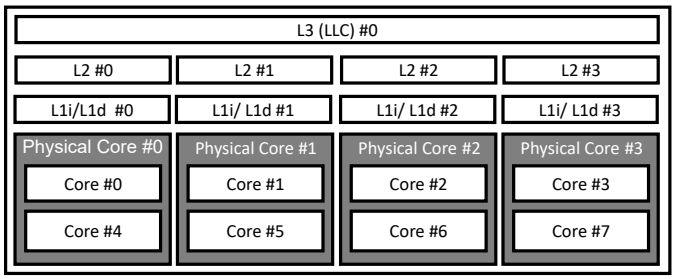

Fig. 4. CPU architecture of the vRAN shared computing platform.

As shown in Fig. 2, the SMO (4) and the mobile core functions (5) are deployed in separated computing nodes from the vBSs. The near-RT RIC is co-located with the vBSs, while the SMO hosts the non-RT RIC and the functionalities to manage and orchestrate the O-Cloud infrastructure. We use a custom version of the non-RT RIC and the near-RT RIC.

As the SMO hosts the functions to orchestrate and manage the O-Cloud infrastructure using the O1 and O2 interfaces, it allows us to set up different virtual networks and start, stop, and remove a dynamic number of vBSs configuring the resources, such as the computing time and the computing cores that shall be allocated to each instance. Furthermore, it can start and stop UEs and the mobile core. We support different testbed scenarios that require ad-hoc solutions to orchestrate the different entities. We develop a set of different functions using the `python`'s `Docker` library for this purpose. We also set up `Docker` daemon instances in the different hosts to retrieve and enforce SMO policies.

Besides, the SMO allows us to define and enforce new configuration policies via the non-RT RIC, which forwards the configuration policies to the near-RT RIC. In turn, the near-RT RIC enforces them in the vBSs. Also, the SMO features an AI/ML engine to support different rApps. The O-RAN RIC control interfaces A1 and E2 are implemented using the `ZMQ` message library. The vRAN orchestration algorithms under test are deployed using the AI/ML engine. This entity has access to a time-series database (8) to retrieve the monitoring metrics.

We use a containerized version of srsEPC [13] to emulate mobile core functionalities. srsEPC is deployed into a separate host reachable by the Edge application server and attached UEs. We connect the vBS to the mobile core using `Docker`'s host networking.

Finally, the operator dashboard (9) is a custom `python` framework that allows us to interact with the SMO. The operator dashboard enables the configuration of the experimental scenario, including traffic and SNR patterns, the use of an AI service located at the edge server, the number of active vBS and UEs, and the vRAN algorithm to be tested.

C. Traffic Generators and Other Applications

Our testbed has an edge application server (6) enabled with a NVIDIA GeForce RTX 2080 Ti GPU. In some scenarios, we use this server as a source for downlink (DL) traffic and a sink for uplink (UL) traffic, using `MGEN` for this

purpose. In other scenarios, this server hosts edge AI services. In our experiments, we select an object recognition service due to its popularity in computer vision applications (e.g., vehicle navigation, surveillance systems, mobile health, etc.) and high resource demand (GPU processing is required). In particular, we deployed `detectron2`, an open-source object recognition software. In our experiments with the edge service, the UE sends an image from the well-known COCO dataset, and the server replies with the bounding boxes and labels computed by `detectron2`. Both the traffic generators and the edge AI services are deployed using Docker containers.

IV. Metrics and Data Storage

To gather monitoring metrics from the vRAN platform and the O-Cloud, we use an O-RAN compliant monitoring system. The near-RT RIC subscribes to the O-RAN components deployed so that it retrieves the different radio metrics through the E2 interface [14]. Afterward, the near-RT RIC passes the data using the A1 interface to the non-RT RIC. We developed an rAPP to push data coming from the different vBS into the time-series database. Moreover, the SMO can set up performance management (PM) jobs to gather metrics from the O-Cloud platform, mobile core, and edge server. We use `Telegraf` and its `file` extension as a metric agent collector to gather the data from all the PM jobs and send it to the time-series database periodically. To ease the final processing of multi-host data sources, we keep clock synchronization of all hosts by using the Precision Time Protocol (PTP). To store the monitoring metrics database, we use `InfluxDB` time-series database. We also use `Grafana` to visualize data in real-time. In the following, we present a complete description of the metrics that can be collected from our testbed.

A. Computing Metrics

The computing utilization for the vBS Docker instances deployed in the computing pool can be gathered by using a PM job that periodically reads the information in `/proc` filesystem (for each thread and in each container), and returns the computing utilization for each computing core in use. The scripts save the information to a JSON file, which can be easily read and processed by Telegraf, enabling xApps and rApps access to this information. Furthermore, we also use the kernel tool `perf` to measure low-level metrics for each container, such as the number of cache misses, the number of core cycles, and the number of instructions.

B. srsRAN Metrics

We collect metrics from all the srsRAN software instances (UEs and vBSs). In the case of the vBSs, we enhance srsRAN by adding the E2 interface allowing the near-RT RIC to subscribe and periodically receive monitoring information from the different layers of the protocol stack, such as the SNR, the uplink and downlink MCS, or the traffic demand for both directions, as well as the uplink decoding time and the subframe time processing. In the case of the UEs, we modified srsRAN to save standard metrics into a JSON file to be read by `Telegraf`.

C. Per-flow Metrics

We gather per-flow metrics of the different traffic generated/received by UEs to/from the application server by using IP tables packet and byte counters. Upon starting a new UE and its traffic generator or application instance, we add two new tables to each container's IP tables, namely `TRAFFIC_ACCT_IN` and `TRAFFIC_ACCT_OUT`, to track traffic into the `INPUT` and `OUTPUT` directions. We add dedicated rules to match the IP addresses of the UE and the application and obtain the cumulative count of packets and bytes hitting these rules, saving them into a file that is periodically read by `Telegraf`.

D. Energy Consumption Metrics

We use software tools and an external digital power meter (7) to measure the energy consumption of different testbed components. In particular, for the software energy measurements of vBSs, we use Intel's Running Average Power Limit (RAPL) functionality using the Linux tool `turbostat`. RAPL estimates the power consumed by the CPU by using hardware performance counters and I/O models. Similarly, we obtain the GPU power consumption using the NVIDIA driver via `nvidia-smi`.

Note that software measurements only consider the main processing unit (CPU or GPU). In contrast, hardware measurements capture the entire platform's power (e.g., CPU, GPU, motherboard, RAM memory, etc.) and the radio head. We use the digital power meter GW-Instek GPM-8213 along with the GW-Instek Measuring adapter GPM-001 to retrieve this data. These measurements are collected by the edge application server via an SCPI interface and saved into a file to be read by `Telegraf`.

E. Radio Control Policies

The vRAN orchestration algorithms can enforce different radio policies on vBSs. As shown in related works using this testbed [2], [5], [4], [15], [7], the use of different radio policies is fundamental, for example, to balance energy consumption and performance or to adapt to the available computing resources. We use the E2 O-RAN interface to control the following radio parameters dynamically:

- Modulation and Coding Scheme: upper-bound and fixed values. This radio policy is used in [2], [4], [5], [7] to set the available computing resources.
- Transmission Gain: to evaluate different SNR patterns or to save energy.
- Airtime (UL and DL Physical Resource Blocks): We configure the maximum number of radio blocks per subframe on uplink and downlink directions, which modifies the ratio of used radio resources.

V. Data Sets

In this section, we describe the organization and metrics of three datasets collected with our testbed and saved in CSV format. The datasets are available on the IEEE DataPort portal [6].

COMPUTING DATASET			ENERGY DATASET			APPLICATION DATASET		
Configuration Parameters			Configuration Parameters			Configuration Parameters		
Column	**Label**	**Description**	**Column**	**Label**	**Description**	**Column**	**Label**	**Description**
1	*mcs_dl_i*	vBS i DL MCS	2	BW	Bandwidth	3	BW	Bandwidth
2	*mcs_ul_i*	vBS i UL MCS	5(6)	traffic_load_dl(ul)	DL(UL) load	4	img_resolution	Image size
3	*dl_kbps_i*	vBS i DL load	7(8)	txgain_dl(ul)	TX gain	5	airtime_ratio	airtime alloc.
4	*ul_kbps_i*	vBS i UL load	9(10)	selected_mcs_dl(ul)	DL(UL) MCS alloc.	6	gpu_power	GPU alloc.
5	*cpu_set_i*	vBS i CPU set	11(12)	selected_airtime_dl(ul)	DL(UL) airtime alloc.			
Measurements			Measurements			Measurements		
Column	**Label**	**Description**	**Column**	**Label**	**Description**	**Column**	**Label**	**Description**
6-13	*cpu_i*	Avg. CPU usage	23(24)	thr_dl(ul)	Avg. DL(UL) throughput	7	av_end2end_delay	Avg. delay
14	*explode*	Successful?	25(26)	bler_dl(ul)	Avg. DL(UL) Block Error Rate	12-17	AP(1-6)	Avg. Precision
			28	pm_power	Avg. HW power	18-23	AR(1-6)	Avg. Recall
			29	pm_var	Var. HW power	24	powermeter_av	Avg. HW power
			30	pm_median	Median HW power	25	powermeter_var	Var. HW power
			31	n_pm	Nr. of HW power samples	26	powermeter_median	Median HW power
			32	rapl_power	Avg. SW power	27	rapl_av	Avg. SW power
			33	rapl_var	Var. SW power	28	rapl_var	Var. SW power
			34	n_rapl	Nr. of SW power samples	29	gpu_av	Avg. GPU power
						30	gpu_var	Var. GPU power

TABLE I
RELEVANT FIELDS IN COMPUTING, ENERGY, AND APPLICATION DATASETS [6].

A. Computing Dataset Description

This dataset relates to the research activities published in [2] and considers the instantiation of a different number of vBSs over the same computing platform. With reference to Fig. 2, we adopted the components (1), (2), (3), (4), (5), (6), (8) and (9). The vBSs are instantiated in specific CPU core sets with different time-sharing allocations. Each vBS has an associated context, composed of the traffic demands and statistics about the used MCS for both UL and DL. We remark that different network parameters (e.g., the MCS index) have impacts on the CPU load, mainly due to coding/decoding workloads. We run a 20-second experiment for each row in the dataset with a specific context, and evaluate the impact (and cross-interference) of the processing workload across the running instances. The resulting per-core CPU utilization is the average of the samples collected every 200 ms.

We collected two sets of data. The measurements in *datasets_unpinned* directory consider the default Linux CPU scheduler policy. It allocates CPU resources in an unrestricted manner and, therefore, the workloads of different vBSs share CPU cores. We consider heterogeneous deployment cases, spawning from one to five concurrent vBS instances. The measurements in *datasets_pinned* directory are collected with a set of CPU cores dedicated to each vBS instance (i.e., CPU pinning) that provides isolation between vBS workloads. In particular, we deploy two vBS instances and consider two different pinning options: (i) pin one vBS to core 1 and the second vBS to core 2, and (ii) we change the pinning configuration of the second vBS to core 5. In this way, we can compare the computing utilization when L1 and L2 caches are shared or not.

Second, we deploy four vBSs and carry out a similar experiment. In the first case, we pin the vBSs to cores 1, 2, 3, and 4, respectively. In this way, vBSs have the L1 and L2 caches isolated. In the second experiment, we pin them in cores 1, 5, 2, and 6, respectively. In this scenario, there is L1 and L2 cache isolation between the sets of vBSs 1,2 and 3,4, but there is no cache isolation between the vBSs 1 and 2, and 3 and 4. The dataset contains the following metrics. Columns *mcs_dl_i*, *mcs_ul_i*, *dl_kbps_i*, *ul_kbps_i* and *cpu_set_i* define the context of a vBS i, which represent the instantaneous DL MCS index, UL MCS index, the traffic demand in downlink and uplink (in kbps), and the CPU core set configuration. The measurements for the i-th computing core are provided by the column *cpu_i*. Finally, when column *explode* takes the value *True*, it indicates that the traffic demand has not been served correctly, which is correlated to the lack of computational resources. Conversely, when *explode* is set to *False* the traffic is served successfully. Table I (left) summarizes the above.

B. Energy Dataset Description

This dataset used in [4], [7] aims to characterize the power consumption of a vBS. These experiments adopted the same components presented in the previous scenario, with the addition of the power meter (7). The main configuration parameters, shown in Table I (middle), are related to the traffic load, SNR, MCS, and airtime in both DL and UL. Note that, to measure different SNR values, we modify the transmission gain of the USRPs.

The dataset comprises two files. The file *dataset_ul.csv* only considers UL traffic [7], while in *dataset_dlul.csv* considers both concurrent UL and DL traffic loads [4]. Each row corresponds to 1 minute execution of a fixed configuration. The most important metrics in the dataset are shown in the bottom part of Table I. We measure the consumed power via software (RAPL) and hardware (digital power meter), as explained in Sec. IV. We also measure the block error rate (BLER) and the throughput. Other interesting metrics in the dataset are the average decoding time of the uplink transport blocks, the clock speed of the CPU in the computing platform, and the buffer state of the UE and vBS.

C. Application Dataset Description

In this dataset, used in [5], we consider the scenario of a mobile user accessing an AI service running in an edge server, and measure how the joint configuration of the vBS, AI service, and the edge server settings impact the power consumption and service performance. To launch these experiments we deploy

an AI/ML application in (6) shown in Fig. 2 and used (1), (2), (3), (4), (5), (8) and (9).

In this dataset, the configuration parameters include the airtime (*airtime_ratio*), the image resolution (*img_resolution*), which indicates the percentage of the original size of the image, and the GPU speed (*gpu_power*), which indicates the maximum power that the GPU is allowed to dissipate. Thus, the higher the GPU speed, the faster the processing.

For each row in the dataset, 150 images from the COCO dataset are processed by the edge server, which returns the bounding boxes and labels of the objects in the image. For each image, we measure (i) the end-to-end delay that includes the time incurred by a user request (an image) to be delivered to the service, the processing time (GPU delay), the time incurred to reach the user with the reply; (ii) the image processing delay (*imp_proc_delay*) indicates the time to load and resize the images at the user side; (iii) the GPU delay (*gpu_delay*) indicates the delay incurred by the GPU at the edge server; (iv) Number of detected objects (*num_obj*); (v) Average precision in the object recognition task (*AP_per_image*). Moreover, we also include in each row global measurements as a result of averaging across all the images. To measure the global performance of the object recognition service, we also provide several precision and recall values, namely *AP*1-*AP*6 and *AR*1-*AR*6 [5].

Finally, to measure the global power consumption, we provide the columns *rapl_av* and *rapl_var* that measure CPU consumed power using RAPL. Additionally, the power consumed by the GPU-enabled edge server is measured using the power meter (*powermeter_av* and *powermeter_var*) and software (*gpu_av* and *gpu_var*).

VI. Novel Applications

In this section, we describe some examples of novel applications for the presented datasets. Using the *computing dataset*, we plot in Fig. 5, the CPU usage of a shared computing platform as a function of a different number of vBS instances for three different channel quality configurations. Specifically, we observe that the CPU usage does not scale linearly with the number of vBS due to the interference among processes (called the *noisy neighbor problem*), even when the processes are pinned. Moreover, we also observe that the CPU usage also depends on other parameters, such as the channel quality. This motivates the need for predictive ML models that can anticipate the CPU demand given the context and the configuration of the vBS instances. This is of enormous importance as a deficit of computing resources can lead to synchronization loss and drastic network throughput decay.

Concerning the *energy dataset*, this data can be used to fit the linear energy model proposed in [7] or a potentially extended model considering also the DL. Similarly, the *application dataset* can be used to train models of the consumed power of an edge AI service. These energy models can be very useful for the research community, as they allow us to accurately predict the consumed power of a mobile network as a function of its configuration and can be used, for example, to derive novel energy-driven strategies for green networking.

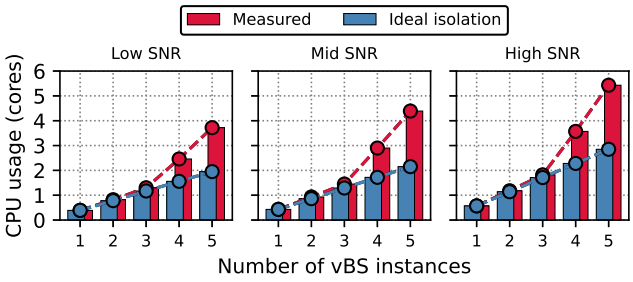

Fig. 5. Computing usage for different SNR configurations and different number of vBSs compared to expected linear increase.

VII. Conclusions

In this paper, we described the setup of an O-RAN compliant testbed, starting from its design principles towards practical implementation and technical aspects, using off-the-shelf networking equipment and virtualization software. Additionally, this paper is accompanied by 3 datasets, collected from our testbed, each one focusing on different scenarios (*i*) *Computing dataset* characterizes the computing usage of vBS instances on shared computing platforms; (*ii*) *Energy dataset* measures the energy consumption of a vBS as a function of a wide range of parameters; and (*iii*) *Application dataset* characterizes the joint impact of the network and an edge service configuration on the energy consumption and performance of the system. We believe these datasets, together with our practical insights, can promote research in this field and foster the development of novel solutions for the efficient sharing and management of radio and computing resources in open radio environments.

References

[1] M. Polese *et al.*, "Understanding O-RAN: Architecture, interfaces, algorithms, security, and research challenges," *IEEE Communications Surveys & Tutorials*, 2023.

[2] J. A. Ayala-Romero *et al.*, "vrAIn: Deep Learning based Orchestration for Computing and Radio Resources in vRANs," *IEEE Transactions on Mobile Computing*, 2020.

[3] A. Manousis *et al.*, "Contention-aware Performance Prediction for Virtualized Network Functions," in *ACM SIGCOMM*, 2020.

[4] J. A. Ayala-Romero *et al.*, "Orchestrating Energy-Efficient vRANs: Bayesian Learning and Experimental Results," *IEEE Transactions on Mobile Computing*, 2021.

[5] ——, "Edgebol: Automating energy-savings for mobile edge ai," in *ACM CoNEXT*, 2021.

[6] J. X. Salvat Lozano *et al.*, "O-RAN experimental evaluation datasets," 2022, Accessed on 09.03.2023. [Online]. Available: https://dx.doi.org/10.21227/64s5-q431

[7] J. A. Ayala-Romero *et al.*, "Experimental Evaluation of Power Consumption in Virtualized Base Stations," in *IEEE ICC*, 2021.

[8] X. Foukas *et al.*, "Concordia: Teaching the 5g vran to share compute," in *ACM SIGCOMM 2021 Conference*, 2021, pp. 580–596.

[9] J. Ding *et al.*, "Agora: Real-time massive mimo baseband processing in software," in *International Conference on Emerging Networking Experiments and Technologies*, 2020, pp. 232–244.

[10] M. Polese *et al.*, "Colo-ran: Developing machine learning-based xapps for open ran closed-loop control on programmable experimental platforms," *IEEE Transactions on Mobile Computing*, 2022.

[11] S. D'Oro *et al.*, "Orchestran: Network automation through orchestrated intelligence in the open ran," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 270–279.

[12] Open RAN Alliance, "O-RAN-WG1-O-RAN Architecture Description – v04.00.00," *Tech. Spec.*, Mar. 2021.

[13] I. Gomez-Miguelez *et al.*, "srsLTE: An open-source platform for LTE evolution and experimentation," in *ACM WiNTECH*, 2016, pp. 25–32.
[14] O-RAN Alliance, "O-RAN Near-Real-time RAN Intelligent Controller Architecture & E2 General Aspects and Principles 2.0," Link, O-RAN Alliance, Technical Specification (TS), 2022.
[15] L. Zanzi *et al.*, "LACO: A Latency-Driven Network Slicing Orchestration in Beyond-5G Networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 1, pp. 667–682, 2021.

Josep Xavier Salvat received his Ph.D. from the Technical University of Kaiserslautern in 2022 and he currently works as a senior research scientist in the 6G Network group at NEC Laboratories Europe, Heidelberg. His research interests lie in the application of machine learning to real-life computer communications systems, including resource allocation and energy efficiency problems.

Jose A. Ayala-Romero received his Ph.D. degree from the Technical University of Cartagena, Spain, in 2019. Currently, he is a senior researcher with the 6G Network group at NEC Laboratories Europe. His research interests include the application of machine learning and reinforcement learning to solve mobile network problems.

Lanfranco Zanzi received his Ph.D. degree from the Technical University of Kaiserslautern (Germany) in 2022. He works as a senior research scientist at NEC Laboratories Europe. His research interests include network virtualization, machine learning, blockchain, and their applicability to 5G and 6G mobile networks in the context of network slicing.

Andres Garcia-Saavedra received his Ph.D. degree from the University Carlos III of Madrid in 2013. Currently, he is a Principal Researcher at NEC Laboratories Europe. His research interests lie in the application of fundamental mathematics to real-life wireless communication systems.

Xavier Costa-Pérez (M'06–SM'18) is Head of 5G/6G R&D at NEC Labs Europe, Scientific Director at i2Cat and Research Professor at ICREA. He received both his M.Sc. and Ph.D. degrees in Telecommunications from the Polytechnic University of Catalonia, Barcelona.