IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 23, 2020, accepted December 6, 2020, date of publication December 10, 2020,
date of current version December 28, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3043876

Testbed for 5G Connected Artificial Intelligence on Virtualized Networks

CLEVERSON VELOSO NAHUM[1], LUCAS DE NÓVOA MARTINS PINTO[1], VIRGÍNIA BRIOSO TAVARES[1], PEDRO BATISTA[3], SILVIA LINS[2], NEIVA LINDER[3], AND ALDEBARO KLAUTAU[1], (Senior Member, IEEE)

[1]LASSE - 5G & IoT Research Group, Federal University of Pará, Belém 66075-750, Brazil
[2]Ericsson Research, Rodovia Engenheiro Ermênio de Oliveira Penteado, Indaiatuba 13337-300, Brazil
[3]Ericsson Research, 164 80 Stockholm, Sweden

Corresponding author: Cleverson Veloso Nahum (cleversonahum@ufpa.br)

This work was supported in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and in part by the Innovation Center, Ericsson Telecomunicações S.A., Brazil.

ABSTRACT The fifth-generation (5G) cellular networks incorporate a large variety of technologies in order to address very distinct use cases. Assessing these technologies and investigating future alternatives is complicated when one relies only on simulators. 5G testbeds are an important alternative to simulators and many have been recently described, emphasizing aspects such as cloud functionalities, management and orchestration. This work presents a 5G mobile network testbed with a virtualized and orchestrated structure using containers, which focuses on integration to artificial intelligence (AI) applications. The presented testbed uses open-source technologies to deploy and orchestrate the virtual network functions (VNFs) to flexibly create various mobile network scenarios, with distinct fronthaul and backhaul topologies. Distinctive features of the testbed are its relatively low cost and the support to using AI for optimizing the network performance. The paper explains how to deploy the testbed structure and reproduce the presented results with the provided code. AI-based radio access network (RAN) slicing and VNF placement are used as examples of the testbed capabilities.

INDEX TERMS Artificial intelligence, machine learning, virtualized mobile network, testbed, 5G.

I. INTRODUCTION

The fifth-generation (5G) cellular network addresses a variety of usage scenarios, including enhanced Mobile Broadband (eMBB), Ultra Reliable Low Latency Communications (URLLC) and massive Machine Type Communications (mMTC). In order to meet very distinct requirements, 5G networks rely on a flexible New Radio (NR) interface and several virtualization technologies, such as Network Function Virtualization (NFV), Software-defined networking (SDN) and Software Defined RAN (SD-RAN). Using virtualization, a better use of computational and network resources is achieved, with the resources being allocated according to the applications' requirements, enabling efficient strategies such as network *slicing* [1].

All the flexibility aimed by 5G to attend the large number of different scenarios leads to an increase in the complexity in the network management. Artificial Intelligence (AI) tools compose an important part of the strategy to automatize the deployment and configurations of parameters in a 5G network, aiming at achieving, for instance, self-organizing networks (SON) [2] and zero touch networks [3].

AI applied for mobile networks [4] is a key enabler to manage not only 5G but also beyond 5G (B5G) networks. The existing 5G mobile networks (3GPP Releases 15 and 16) have no sufficient flexibility and intelligence to fulfill all requirements for the three main use cases (eMBB, URLCC and MTC) yet [5]. Over the years, the networks were not appropriately designed to accommodate AI-oriented tasks such as data collection, processing, and output distribution. Current systems are typically designed to deliver content, without considering the characteristics from each application requesting network resources [6]. When properly integrated, AI can leverage the efficiency of modern and sophisticated networks by using, e. g., *big data* techniques.

Not only the amount of data that is transmitted, but also the one used to manage and operate mobile networks is huge. This fact is motivating the flourish of Machine Learning (ML)

The associate editor coordinating the review of this manuscript and approving it for publication was Qichun Zhang.

for communications. While AI encompasses planning, expert systems and other topics, especially due to recent advances in deep learning, ML has gained considerable momentum in areas such as computational vision and also communications [7]. The main characteristic of ML when contrasted to other AI techniques is that the designed models are based on experience and learned automatically from data [8], which is abundant in mobile communications. Both Radio Access Network (RAN) and core network have a large amount of information about the mobile network, user equipments (UEs), and applications running, which can be extremely useful to learn complex patterns and improve network operations.

Therefore, several standardization bodies are working to incorporate AI / ML in 5G and B5G mobile networks, such as the 3rd Generation Partnership Project (3GPP), European Telecommunications Standards Institute (ETSI) and International Telecommunication Union Telecommunication Standardization Sector (ITU-T). For instance, ITU-T has the Focus Group on Machine Learning for Future Networks including 5G (FG-ML5G), which created a ML architecture to be integrated to the future mobile networks, providing a common nomenclature for ML-related mechanisms and keeping interoperability with other networking systems [9]. Different from the ITU-T studies, 3GPP and ETSI focus on centralized data collection and data analytics solutions. 3GPP is discussing, for instance, architectural requirements for enabling AI within the scope of Release 17 under study [10]. ETSI actively studies the integration of AI into networks through groups developing Experiential Networked Intelligence (ENI) and Zero-touch network and Service Management (ZSM) [11].

The mentioned studies in standardization bodies, complemented by efforts within the industry and academia, dictates the pace of development of APIs and codebases for AI-enabled communications [12]. In order to assess and integrate the newly developed AI algorithms, testbeds are a sensible alternative to simulators, as widely discussed in the scope of other communications paradigms such as mesh networks [13]. Testbeds are essential for developing and assessing network technologies since they provide more realistic scenarios regarding simulation and the solutions developed on testbeds have a shorter path between the research stage and products [13]. The importance of testbeds increase in 5G/B5G due to the sophisticated network features which are hard to reproduce in their totality using simulations.

This paper describes the Connected AI (CAI) testbed structure to build flexible and realistic scenarios with different network topologies for 5G and quickly deploy them. A container-orchestrator makes the deployments of the applications to create a mobile network interconnected to virtualized OpenFlow switches, an SDN controller, and a RAN controller. The testbed is built with open-source software and Virtual Network Functions (VNFs) to facilitate the deployment and research of different network scenarios without the need to use specialized hardware. This paper presents two use cases, one for RAN slicing and another for the placement of VNFs according to application requirements. The source code is made available to allow replicating these two experiments. The experiments were designed to be easy to follow, and decrease the learning curve to get started with AI techniques applied to 5G using the CAI testbed.

The contributions of this paper are the following:

- It presents a low cost testbed that emphasizes AI integration using Kubeflow to the ML workloads associated to the management of 5G networks. The proposed testbed avoids the complexity of Management and Orchestration (MANO) agents to provide cloud services and focuses in AI applied to RAN and transport network, including fronthaul and backhaul.
- The paper describes two use cases that explain how AI can be used in the testbed, leveraging collected information and applying AI decisions to optimize the network.
- The experiments are reproducible, given that the code to deploy the testbed and the two use cases applications are made publicly available.

This paper is organized as follows. Section II describes the 5G testbed proposals in academy and industry, emphasizing their goals and applications. This helps to contextualize the CAI testbed. Section III describes the CAI testbed structure and main features. Section IV describes two use cases to apply AI techniques using the testbed and their respective results. Section V explains the open issues, next steps and concludes the paper.

II. RELATED WORK: TESTBEDS AND THEIR FEATURES

Several 5G testbed implementations have been developed [14]–[17] to demonstrate functionalities and operation of a 5G system. Given the variety of technologies in 5G, to make the following discussion more concrete, special emphasis is placed on slicing and VNF placement, which are the two use cases described in Section IV.

An important feature of 5G networks is the possibility of an infrastructure (InP) to share its physical resources among different Mobile Virtual Network Operators (MVNOs), providing isolation, security, and accomplishing the Service Level Agreements (SLAs) requested by each operator. NFV is essential to make better use of physical resources and making the VNFs allocations along the infrastructure using orchestrators. The network slicing facilitates the MVNO and InP provisions of different services in the network with dedicated resources [1]. With the wide variety of applications and scenarios, it becomes more important to choose the better network parameters and VNF placement to improve the performance of each network slicing without affecting others. ML models fed with network information to choose these parameters and the placement of VNFs in the infrastructure can enhance the network slicing operations, as demonstrated in many works [18]–[20]. Not necessarily involving slicing and NFV placement, a brief review of related works in 5G testbeds is presented next.

A scalable orchestration architecture based on a cross-domain optimization is presented in [14]. It works by providing an abstraction of physical resources for the orchestration level, allowing to implement network slices according to network applications. Only RAN and transportation resources are allocated in the use cases presented in this work, where RAN instances are deployed using VNFs and the transportation resources are implemented in optical switches, which are integrated into a SDN controller managing the routes and communication among RAN VNFs.

The authors in [15], [16] discuss a testbed implementation with an SDN/NFV packet/optical transport network and edge/core cloud platform for end-to-end 5G and Internet of Things (IoT) services deployed with open-source software and Commercial Off The Shelf (COTS) hardware. The work in [17] implements a similar testbed using an SDN controller managing an optical/wireless fronthaul. These works focus on the VNFs allocations and SDN architectures for mobile networks mixing open-source and commercial software. They demand relatively robust physical infrastructures due to the requirements for their testbed implementation. The authors do not provide code or more detailed descriptions about how to deploy and adapt these testbeds to enable their adoption by other groups, especially those with limited infrastructure resources. Also, network slicing and machine learning integrations are not explored.

Some other testbeds focus in MANO implementation to offer a partial or full end-to-end network slicing over RAN, transport network (TN) and core network [21]–[23]. In [21] there is a focus on flexibility and scalability to service provisioning, having been applied to certain use cases, such as critical e-health, vehicle-to-everything communication for intelligent transportation systems and multi-service management for smart cities. The works in [22], [23] provide independent and customizable end-to-end slices, where [23] focuses in an integration between OpenAirInterface (OAI) and M-CORD, enabling the implementation of different procedures to deploy 4G network over M-CORD.

It is convenient to briefly describe the pros and cons of integrating MANO into 5G testbeds, given that the proposed CAI favors AI tools and does not adopt MANO for the sake of reducing the infrastructure requirements and cost. The main MANO project implementations are the OSM and ONAP [24], where ONAP needs a higher computational resource availability and OSM has less stringent requirements. Neither of them support Virtual Infrastructure Manager (VIM) with containerized VNFs, so the VNFs are deployed into virtual machines, which spend more computational resources than containerized applications [25].

The advantage of using MANO implementations are related to performing efficient resource management and orchestration, and the management of network slicing lifecycles. The MANO implementation can be avoided in researches that are not focused in the end-to-end network slicing or in aspects as the management of their lifecycles. For example, researches on RAN aspects such as slicing-aware radio resource management (RRM) or network slicing in the TN can be made without considering MANO features, enabling the evaluation of less complex but diverse scenarios that are focused on specific applications.

TABLE 1. 5G Testbed's comparison.

Testbed	SDN	NFV	MANO	Slicing	Open-source	AI workloads	TN
[14]	Yes	Yes	No	Partial	No	No	Physical
[15]	Yes	Yes	No	Partial	No	No	Physical
[16]	Yes	Yes	No	Partial	No	No	Physical
[17]	Yes	Yes	No	Partial	No	No	Physical
[21]	Yes	Yes	Yes	End-to-end	Yes	No	Physical/Virtualized
[22]	Yes	Yes	Yes	End-to-end	Yes	No	Physical/Virtualized
[23]	Yes	Yes	Yes	End-to-end	Yes	No	Physical/Virtualized
CAI	Yes	Yes	No	Partial	Yes	Yes	Emulated

Another important aspect of the mentioned testbeds [14]–[17], [21]–[23] is that the AI integration to enable researches is not emphasized. For instance, there are no descriptive guidelines on how to use the controllers to get information about the network and apply AI techniques. Also, there is no structure to orchestrate the AI workloads over the physical structure.

Table 1 shows a comparison of testbed features, including the proposed CAI testbed. Despite the partial network slicing and absence of MANO, the CAI testbed presents an orchestration to AI workloads to facilitate and turn more intuitive the deployment of AI agents into the testbed. It also presents an emulated TN that enables the deployment of any network topology on fronthaul and backhaul, without the need to have access to actual transport network topologies. This facilitates the exploration of different scenarios with reduced resources and cost. The next section provides more details about the proposed CAI.

III. CAI: THE IMPLEMENTED CONNECTED AI TESTBED

The CAI testbed enables the building of flexible and realistic AI-based scenarios with different network topologies for 5G and quickly deploy and assess them. The SDN and RAN controllers work as information sources about the network. They also work as agents to dynamically change the mobile and the computer network. An AI agent performs different actions in the testbed according to the application, using the information provided by SDN and RAN controllers to train and execute in test stage its neural networks. The ML workloads are orchestrated along the cluster to provide the AI agent processes. All the code to set up the testbed and execute the use cases, which will be explained in the following sections, are available on Github.[1] The next paragraphs detail each component of the testbed.

A. ORCHESTRATION

The container-orchestrator has the responsibility to create a cluster with available machines, interconnect them through a Container Network Interface (CNI), and make the application deployment and orchestration of the cluster nodes. The cluster has two types of nodes: the master and the worker nodes, where the master executes the container-orchestrator com-

[1] https://github.com/lasseufpa/connected-ai-testbed

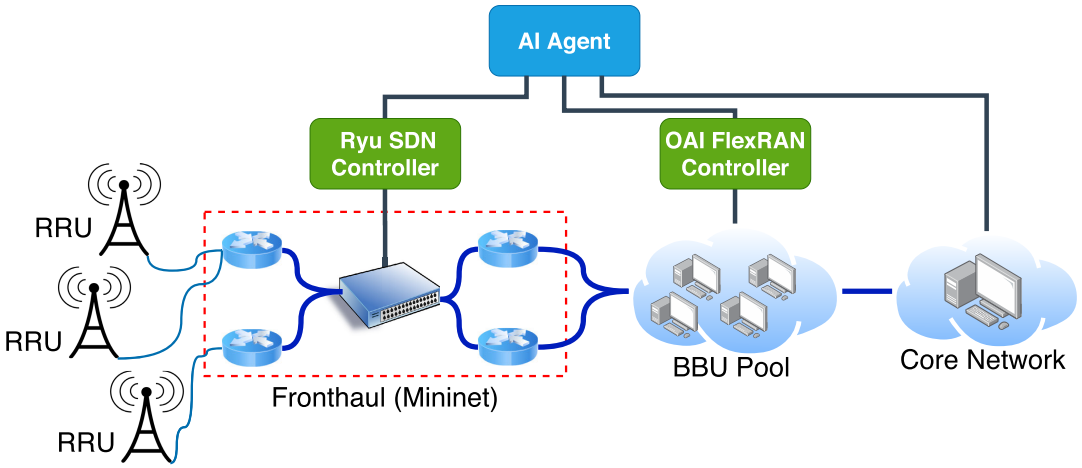

FIGURE 1. Big picture of the proposed testbed. A Cloud RAN (C-RAN) scenario with a virtualized fronthaul using Mininet is deployed. Both an SDN and a RAN controller are used to get information and apply changes in the network.

mands and makes the application deployments. Furthermore, the worker nodes are responsible for executing the containers from applications requested by the master nodes. The CNI has the responsibility to create and manage the network to connect the containers and nodes from the container-orchestrator cluster, enabling the communications among the applications deployed through a virtual network created by the CNI. CAI uses Kubernetes [26] as the container-orchestrator, and Calico [27] as the CNI.

Fig. 1 shows the testbed modules interconnection considering a C-RAN scenario, which is one of the scenarios available in CAI. The figure depicts a virtualized fronthaul deploying one specific topology, which can be redefined with flexibility via scripts. Both SDN and OAI FlexRAN controllers work as an information source to the AI agent.

Fig. 2 depicts the NFV architecture implemented in the testbed. It is similar to an NFV achitecture implemented with MANO, but simpler and with less functionalities. The NFV Infrasructure (NFVI) is composed of the hardware resources which are represented in the testbed by the computers of the Kubernetes cluster. The virtual computer, storage and network are allocated using Kubernetes and Docker through the virtualization layer, which allocates the virtualized components in the hardware resources. The VNFs are represented by the containerized elements that compose the mobile network structure.

Instead of a MANO implementation, an orchestrator of network scenarios is implemented using Kubernetes deployment files, which are much less complex and have less functionalities than MANO. However, they provide a lightweight and simple method to deploy different network mobile architectures along the testbed, avoiding the complexity of MANO implementations in scenarios which do not focus on it. The implemented orchestrator defines the places where the VNFs are allocated in the NFVI structure.

An example of a C-RAN architecture function distribution is represented in Fig. 3, where mobile network functions are distributed along three machines and one of them cre-

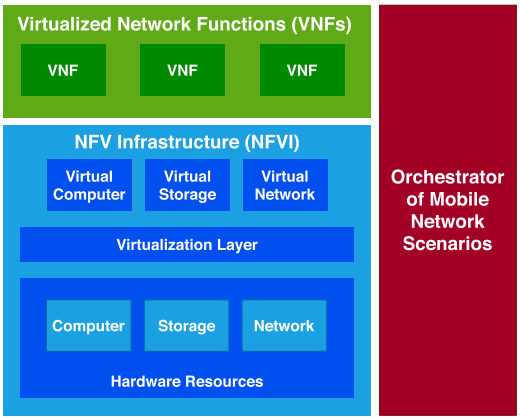

FIGURE 2. NFV arquitecture implemented in the testbed.

ates a virtualized fronthaul and backhaul emulated using Mininet. A USRP B210 generates the radio frequency (RF) signals to a Galaxy S4 smartphone representing the UE. The cluster is composed of 4 machines with Intel Core i5-7500 CPU@3.4 GHz, 8 GB of RAM DDR4 memory, using an operational system Ubuntu 18.04 with low-latency kernel. The testbed does not need all these machines since all VNF functions can be allocated in the same computer using the ochestrator. The number of machines required in the testbed depends of the scenario to be deployed. Since a C-RAN scenario may encompass three locations (antenna, edge and cloud), it is assumed that the mobile functions need to be distributed along three machines, to enable the traffic to be forwarded to the Mininet machine that emulates the backhaul and fronthaul.

The OAI platform [28] has 4/5G modules for both the core network and RAN implementation but only RAN modules for 4G were virtualized, considering a monolithic evolved node B (eNB) and C-RAN [29] scenario through containers implementation using Docker [30]. OAI 5G RAN implementation is not stable yet, so in this work we consider only 4G RAN implementation. The testbed can be easily adapted

to work with OAI 5G RAN as soon as its implementation finishes due to the virtualization structure with containers. For the core network, CAI adopts containers using the Free5GC implementation [31] that supports the 5G core network architecture.

Orchestrator deployment files define the OAI and Free5GC modules deployment and the cluster nodes, following the specifications to each module and assigning them to nodes that fulfill the requirements for each implementation. OAI and Free5GC modules have different machine and configuration requirements, so there is a need to verify which node can satisfy them to be able to deploy these modules. The Kubernetes assign labels to the cluster machines to define which machines can execute each module as defined in the cluster setup. When the orchestrator deploys a container in the cluster, it needs to choose a node that satisfies the minimum requirements to deploy the container.

Each deployment of the testbed has its configuration and machine requirements, so in the deployment files there are definitions of the requirements for each one. For instance, to deploy an remote radio unit (RRU), the machine needs to have a USRP connected. Hence, the Kubernetes verifies which machines have a label identifying that the node has a USRP board connected, and then chooses one to allocate the RRU container. With this approach, the testbed can add more machines to its infrastructure and the orchestrator will be able to assign the deployments to the nodes that can execute the container. Another configuration possible is to define labels as antenna, edge, and cloud to define the mobile network locations to each of the cluster machines. Furthermore, Kubernetes will allocate the modules in their defined locations chosen by the user. Both modes are enabled in the testbed.

OAI can use software defined radio (SDR) boards such as the USRP to generate and transmit LTE signals over the air to UEs, but for scenarios that require a high number of UEs connected to the network, this approach can increase the costs. OAI offers an option to emulate UE and RF signals using the computer instead of using SDR boards and real UEs. This module runs both UEs and eNB/RRU.

With the emulation option, a great number of UEs can be emulated per computer, giving more flexibility to deploy different scenarios. A network interface into the Linux system is created for each emulated UE, so any Linux command can be executed to create traffic into the mobile network, e.g., iperf tools and ping command. The testbed has containers to emulate UEs and baseband functions to enable the use of emulated UEs in the testbed. The testbed supports both execution using USRP to connect real UEs or using emulated ones.

B. VIRTUALIZED FRONTHAUL/BACKHAUL AND SDN CONTROLLER

In a C-RAN deployment using RRU and baseband unit (BBU), the fronthaul is the link connecting these modules. Similarly, the backhaul is the link connecting the BBU or the eNB to the core network. OAI implements both fronthaul and backhaul over an Ethernet connection. Each scenario of mobile network deployment presents different topologies to the fronthaul and backhaul represented by physical components built to enable the transport network, e.g., routers and switches. The testbed implements both the fronthaul and backhaul using a virtual network deployment based on Mininet [32].

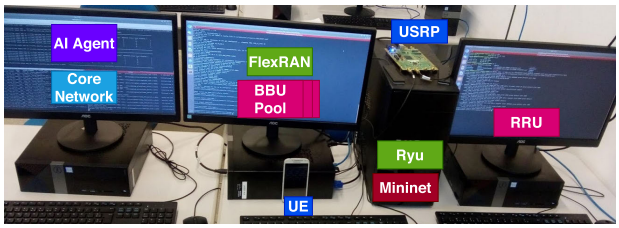

FIGURE 3. Testbed working at LASSE - UFPA lab using a C-RAN architecture.

The use of Mininet gives more flexibility to deploy the fronthaul and backhaul since any network topology can be defined using Mininet scripts, which set virtualized routers and switches to compose the network. The virtualized routers/switches created with Mininet cannot communicate directly to networks outside the virtualization since it can see only the links inside Mininet virtualization as defined in the topology script. To create a link between the host machines and Mininet hosts, we created virtual Ethernet devices (VETHs) linking them, enabling the traffic forwarding between two machines through the Mininet topology. Therefore, the traffic of the fronthaul/backhaul can be forward through Mininet virtualized network.

Figure 4 illustrates how the RRU and BBU machines forward their fronthaul traffic through the virtualized topology instantiated using Mininet. This figure does not show the SDN controller or other Mininet hosts (routers and switches) because it focuses on how machine interfaces are connected to the virtual network to facilitate understanding. Each machine has an Ethernet interface (ETH) connected to the same network. The Mininet hosts can communicate with each other through virtual interfaces (VIs) and virtual switches defined in Mininet script topology, but they are not able to communicate with the ETHs from machines, so the fronthaul traffic is not able to be forwarded through Mininet topology using the default deployment. Creating the VETHs, machine interfaces, and Mininet hosts can establish a communication link, enabling the fronthaul traffic forwarding between the RRU and BBU machines.

Using Mininet to deploy the fronthaul increases the flexibility to build different scenarios since just a script configuration is sufficient to change the topology, packets delay in the VIs and connecting an SDN controller to the virtual switches deployed. Also, it is possible to execute all SDN functions as routers management and link monitoring. The SDN controller acts as an information source about the network states and as an agent to deploy online changes in the structure of how

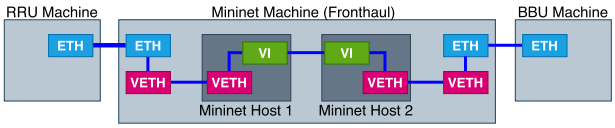

FIGURE 4. **Mininet integration to virtualize the backhaul or the fronthaul.**

the Mininet hosts communicate with each other. The same structure from virtualized fronthaul can be deployed for the backhaul. In this work, we use the Ryu implementation to work as the SDN controller [33] since it is simple to configure and implement algorithms.

C. RAN PROGRAMMABILITY

RAN programmability, also called SD-RAN, works as an abstraction of the RAN resources and by providing an API which enables the Service Orchestrator entity to dinamically manage the RAN resources to provide information about the mobile network [1]. The FlexRAN protocol [34] has defined and implemented an SD-RAN architecture integrated with OAI platform which incorporates an API to separate control and data planes for the mobile RAN.

This architecture has a master controller represented by the FlexRAN controller in Fig. 1 and a FlexRAN agent which corresponds to the OAI eNB instances. In a C-RAN scenario the FlexRAN agent is located in the OAI BBU instances, also represented in Fig. 1. The agents can act as local controllers with a limited network view and handle the functions delegated by the master or being coordinated by the master controller. The FlexRAN Agent API makes the control and data plane separation allowing the control data to be managed by the FlexRAN controller and the eNB data plane on the other side.

FlexRAN APIs enable the development of applications related to the control and management of the RAN resources [34], e.g., schedulers, interference, and mobility manager. Moreover, applications related to improvements in the use of RAN resources to make more sophisticated decisions [34], such as RAN slicing and adaptative video streaming based on channel quality. FlexRAN does not control the flows in the wired domain, so it does not support the management of routes, packets filtering, and other functions related to the computer network domain (routers and switches). These functions can be executed using the Ryu controller.

D. AI INTEGRATION

The mentioned FG-ML5G group associated to the ITU-T, has defined a logical interoperable architecture for future networks, which incorporates a ML overlay that operates on the top of any specified underlay network technology [9]. For instance, based on this architecture, the authors in [6] discussed ML in the context of IEEE 802.11 Wireless Local Area Networks (WLANs). This architecture facilitates deploying ML applications in different network scenarios and is adopted in the CAI testbed. More specifically, the FG-ML5G defined high-level architectural components to integrate ML to the network and a process pipeline. Fig. 5 depicts these components, the pipeline and their respective mapping into the CAI testbed components.

The source (SRC) component is the source of the data that works as input to the ML pipeline. Both the Ryu and FlexRAN controller and core network can provide information through their APIs. The processes "1" in Fig. 5 represent the source of information. The collector (C) component collects data from all sources of the network, as the FlexRAN and Ryu controller and the core network. The collector can choose which source of information to use, following the application deployed in the AI agent. So, if the application requires only information from controllers and not from the core network, it can collect only this information.

The preprocessor (PP) component is responsible for cleaning and aggregating data or performing other preprocessing needed to make data suitable to the ML model. After the data preprocessing, it can feed the model (M) component, which is the ML model to train and execute the neural networks using any available ML library, such as TensorFlow [35] and PyTorch [36]. The policy (P) component enables the application of policies to the output of the model, e.g., it can be used to define when the output should be sent by the AI agent in a live testbed since there are actions that have specific events to be applied in the network.

The distributor (D) component sends the model outputs to the corresponding SINK components which is responsible for promoting actions in the controllers and core network (represented with process "2" in Fig. 5). Using this testbed architecture, information can be provided by controllers and core network to an AI agent which returns actions to these entities to improve the network functions.

The CAI testbed orchestrates the ML workloads of the AI agent using the Kubeflow tool [37]. Kubeflow works integrated to Kubernetes to orchestrate the ML functions along the cluster machines. Kubeflow enables the use of pipelines to define the steps of ML processing. The testbed implements each of the AI agent components described in Fig. 5 as pipeline components, distributed along the testbed in according to the configuration defined. For simplicity, CAI deploys the AI agent at the cloud location (with the core network) due to high resources availability at cloud in real scenarios. But functions could be distributed in other machines in accordance with the defined scenario, as will be discussed in the next section.

IV. CAI TESTBED USE CASES

The proposed testbed structure can be applied in different scenarios and be integrated into different AI agents to improve the mobile network functions. The next subsections provide two use cases of the AI agent. The first is a RAN slicing application to monitor the percentage of resource blocks (RBs) allocated for each requested slice. The second application is an AI agent to define the placement of VNFs along with the testbed's cluster machines.

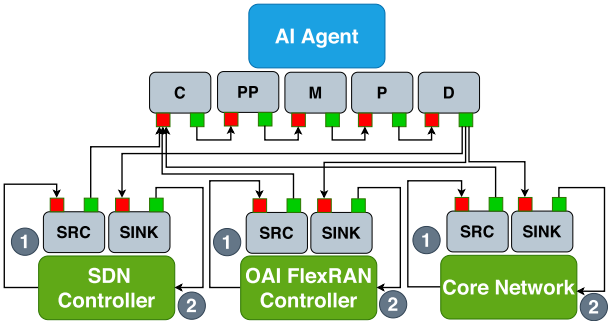

FIGURE 5. Components from the ITU FG-ML5G architecture is applied to the ML implementation of the CAI testbed.

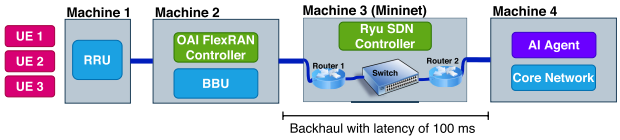

FIGURE 6. RAN slicing scenario with 3 UEs connected to a C-RAN structure with the backhaul virtualized using Mininet.

A. RAN SLICING

In [38], an application for RAN slicing is proposed, utilizing an in-network deep learning to analyze packets in the Packet Gateway (PGW) at the core network and identifying mobile applications. Afterwards, the eNB can apply specific slice configurations for each UE according to requirements associated to the detected application. The work presented in [39] designs and prototypes a network slice solution in a C-RAN architecture that aims to share spectrum among slices efficiently considering their requirements. The authors in [39] use FlexRAN information to identify application requirements and verify whether a service request can be satisfied according to the real-time network state, and communicates the slicing decision. It uses an algorithm for dynamic slicing that estimates the resource allocation satisfaction and shares unused RBs among applications which requires more resources.

Both [38] and [39] consider a monolithic implementation of eNB or a C-RAN architecture implementation where simplified topologies for fronthaul and backhaul are considered and network delay is not accounted for. Link delays from fronthaul and backhaul topology are essential factors for applications that require low-latency and can affect availability and throughput. Using our proposed testbed, more stringent results for slice applications to different fronthaul and backhaul scenarios can be obtained, leveraging the Mininet implementation. Information from Ryu, FlexRAN controllers, and core network can provide useful information to feed ML models and infer possible slice configurations to the services provided by the mobile network.

FlexRAN supports RAN slicing through an API communication that enables to define dedicated RRC/RLC/PDCP/MAC[2] instances for each slice. Moreover, physical resources are strictly dedicated to a specific slice [40], where a percentage of RBs are assigned for each one. FlexRAN also allows defining: if the resources allocated are shared or isolated from other slices, priority, maximum code scheme and schedule algorithm used for each slice. The default configuration assigns all UEs to the same slice, and new slices need to be created before assigning a UE. Slices can be manually defined, or slices manager applications can be connected to FlexRAN API for their creation and control.

We implemented a RAN slicing scenario using FlexRAN and considering eNBs in a C-RAN architecture, where Mininet virtualizes the backhaul to implement a network topology with switches and routers. Fig. 6 shows the scenario with 3 UEs (smartphones) connected to the RRU. A fronthaul composed by a single path connects the RRU to the BBU pool. The BBU pool is connected to the Free5GC core network through a virtualized backhaul using Mininet, containing two routers and one switch connected to the Ryu controller. The total latency added in the backhaul was 100 ms.

In this scenario, the main source of information is the Ryu SDN and the OAI FlexRAN controllers. A customized script makes the Ryu controller provide throughput, link delays and jitter in backhaul link, besides the default information provided by the default Ryu API, e.g., number of packets transmitted and received in each switch port. The FlexRAN provides information about the eNBs, RAN slices and UEs connected, such as the percentage of RBs allocated for each slice, resources isolation and the identification of all UEs connected in a specific eNB.

For our use case, we consider three different RAN slice types: constant bit rate (CBR), minimum bit rate (MBR) and best effort (BE). These types are similar to the ones explored in [41], [42]. Applications that need constant bit rates use the CBR slice, e.g., video conference and voice-over-IP. The MBR slice supports applications that have minimum bit rate requirements but improved performance when offered an increased bit rate, e.g., video streaming services and online games. The BE slice attends applications with no stringent requirements, e.g. Web navigation. Using the FlexRAN API, the three mentioned slices in the eNB were created, where the CBR has the RBs isolated (they cannot be shared with other slices), while MBR and BE can share their resources. The RRU has 3 UEs connected, with the UE 1 using the MBR slice with a minimum throughput of 5 Mbps, UE 2 using the CBR slice with a constant throughput of 10 Mbps and one UE 3 requesting a throughput of 5 Mbps using the BE slice.

Our main motivation is to show how the testbed structure can support ML models by discussing a simple ML algorithm applied to RRM. In this paper, we do not aim at describing or assessing a state-of-art RRM algorithm. The provided source code gives details about all the inputs to the AI agent, which include the number of UEs associated with each slice, isolation of resources, and percentage of RBs for each slice.

[2]RRC: Radio Resources Control, RLC: Radio Link Control, PDCP: Packet Data Convergence Protocol, MAC: Medium Access Control.

TABLE 2. Topology of DNN used for RAN slicing.

Layer type	# input units	# output units	Activation
Dense	15	80	Linear
Dense	80	80	Linear
Dense	80	3	SoftMax

If the model should be optimized, some of these inputs could be eventually discarded. But we tried to incorporate diverse features, in order to illustrate the main point: with a relatively low-cost testbed, one can explore many aspects regarding the application of AI to 5G / B5G networks. In this use case, the outputs of the ML model are the three percentages of RBs to allocate for each proposed slice.

As mentioned, the information from Ryu and FlexRAN controllers is concentrated in the collector C of the AI agent and preprocessed in PP to save in the database only the information to be used as entries of the specific RAN slicing ML model implemented using the PyTorch framework. Based on information from scenario described in Fig. 6, we used a script to generate the database with information from FlexRAN and the Ryu controllers. Each line is labeled with the percentage for each slice using a simple formula provided in the code to make a guarantee of the RBs to CBR slice, and after providing resources to ensure the minimum bit rate to MBR and the remaining RBs to BE slice. The database is interpreted as a table. Each row corresponds to an example, and has a label indicating the desired ML model output, such that one can use supervised learning. Specifically, the label here is the percentage of RBs allocated for each slice type and the problem is posed as regression [43].

Due to their popularity, a deep neural network (DNN) was adopted for this RAN slicing problem. Other learning paradigms could be adopted, such as decisions trees. Table 2 informs the topology of the adopted three-layers DNN. The input layer receives the 15 parameters from the preprocessing in PP, representing the Ryu and FlexRAN information. The output of the network uses a SoftMax activation and has three neurons, which indicate the percentage of RBs for each slice. The RAN slicing use case illustrates the ML model in a closed loop within the network operation. The next paragraph discusses how the DNN output is used to change the RB allocation.

The ML output needs to be adapted to the FlexRAN API format. The policy component P defines that the command should be sent to this API in every 5 seconds. Afterward, the distributor D is responsible for sending the information to the FlexRAN controller API which updates the percentage of RBs for each slice type. Fig. 7 shows the Kubeflow web interface with the pipelines deployed for RAN slicing, implemented in accordance with Fig. 5. This interface provide information about the pipeline deployed as the output of each component and its connections. First, Kubeflow trains the agent, then it deploys pipeline components defined to the AI agent in Fig. 5 and generates the ML output.

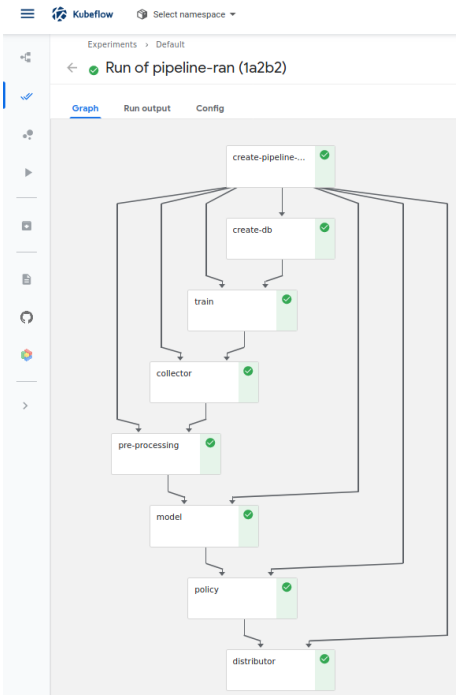

FIGURE 7. Pipeline visualized on Kubeflow webview of the RAN slicing application.

Fig. 8 shows the graphs of UE download throughputs in an eNB using 5 MHz with a maximum download throughput around 18 Mbps. The iperf tool was used to implement a server into the mobile network, and iperf clients were used in the interfaces of emulated UEs. Each UE client requires different amounts of throughputs, as described before. The total duration of the experiment was 1 minute. In the first 20 seconds, the three UEs are active and requesting the maximum throughput reachable. From 20 to 40 seconds, UE 3 is disconnected. After 40 seconds, UE 2 is disconnected, and UE 3 is connected again. The same download traffic was generated to the three UEs and two strategies for RB allocation are contrasted: without using slices versus using the allocation dictated by the trained DNN.

Fig. 8 shows that the DNN provides a better allocation of resources for the UEs. This is expected, given that we are comparing the DNN result with the situation without slicing (instead of using a state-of-art baseline). The goal here is to illustrate the proof-of-concept, with the CAI testbed showing it is capable of incorporating a DNN model that provides differentiation of the traffic according to each slice's requirements. In the situation without slicing (x markers), the scheduler tries to provide a fair division of the radio resources among the connected UEs. For instance, in the first 20 seconds, each UE achieves 6 Mbps. When UE 3 is disconnected, the other 2 equally split the total rate and reach 9 Mbps each. In contrast, considering the AI agent allocation, the requirements specified to CBR and MBR slices are fulfilled in the first 20 seconds. From 20 to 40 seconds, AI agent

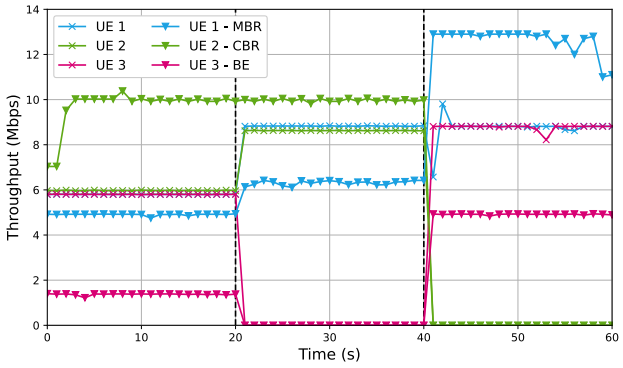

FIGURE 8. Throughput reached for each slice using a AI agent to control the radio resources allocated among the slices.

keeps the bit rate to CBR slice since it just need the constant bit rate and allocate the remaining RBs to the MBR slice. In the last 20 seconds, the AI agent provides more resources to MBR and also fulfill the 5 Mbps requested by the less prioritary slice (BE).

For simplicity, we adopted supervised learning. But CAI is also a convenient testbed to explore RAN slicing and RRM problems tackled with reinforcement learning (RL) methods [44].

B. VNF PLACEMENT

The next paragraphs discuss how the CAI testbed was used in a VNFs placement application. CAI leverages the implementation of mobile network modules as containers, which enable the deployment of different VNFs architectures over the physical structure. For instance, CAI supports the test of different processes of network slicing. The requirements of a requested slice, defined in the SLA, can be processed by a function mapper, which uses the testbed to implement the VNFs in the locations of the network to better fulfill the requirements. Fig. 9 represents different function placements implemented with the CAI testbed through orchestration using Kubernetes.

Fig. 9a represents the well-known C-RAN scenario, which saves deployment costs by having core network functions allocated in the cloud, while the BBU is allocated in an edge facility and RRU near UEs. However, network slices with more strict requirements, such as low latencies and high throughputs may require to allocate VNFs nearer the users. Fig. 9b represents a scenario where a monolithic eNB is deployed near the user, requiring more computational resources but reducing latencies. In contrast, Fig. 9c depicts a scenario where the RRU is allocated near UEs, but some elements from the 5G core network – the access and mobility management function (AMF) in this case – are allocated together with the BBU, which provides lower latencies than the scenario depicted in Fig. 9a. The scenario depicted in Fig. 9d allocates both AMF and user plane function (UPF) at the edge.

Fig. 10 shows the latency perceived by UEs in the different placement scenarios. The elements in set $\mathcal{L} =$

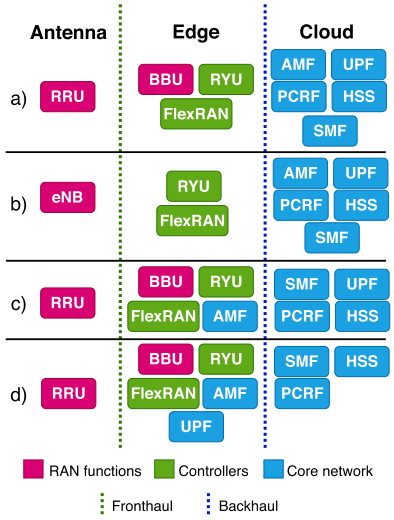

FIGURE 9. VNF placement scenarios allocated along antenna, edge and cloud locations.

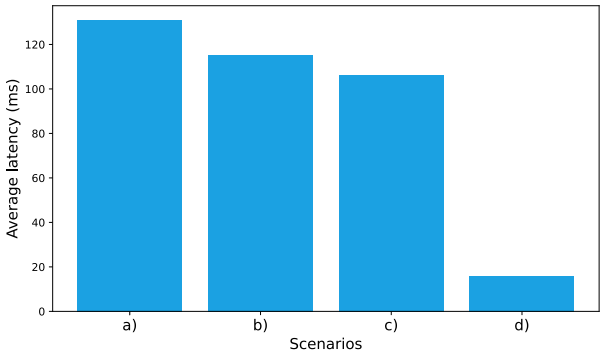

FIGURE 10. Average end-to-end latency for the different placement scenarios in Fig. 9.

{131.1, 115.4, 106.4, 15.8} ms correspond to the average latency values for scenarios a) to d) in Fig. 9, respectively. The scenarios with VNFs located near to users present a lower latency. For instance, the scenario in Fig. 9d presents the lowest latency, as expected, since placing the UPF function in the edge enables the possibility for the UEs to access the packet data network (Internet) via a direct link, avoiding the latency of 100 ms imposed by the backhaul emulated with Mininet.

In this use case, we assume the ML model is capable of indicating which of the four NFV placements in Fig. 9 should be adopted for a given network condition and established SLAs. Again, we do not aim at optimizing the ML model, but illustrating that CAI can actually allocate the NFVs on demand, based on the output of a DNN. To create the training dataset, we collected a variety of input features from the network and also from the key performance indicators (KPIs) corresponding to the requested slice. We then manually determined the ML model output (“correct label”) for each input vector. In a real-life application, such strategy is often unfeasible due to the large number of input features

TABLE 3. ML model layers for VNF placement.

Layer type	# input units	# output units	Activation
Dense	9	80	Linear
Dense	80	80	Linear
Dense	80	4	SoftMax

and eventual lack of an algorithm to define the best output. To speed up this ''manual'' assignment of labels, a script generated the training dataset with the input features and respective labels. For this experiment, a single input feature defines the best output (label): the required maximum latency $\ell_{\max}$. The details are provided in the next paragraphs.

The y-th element of $\mathcal{L}$ is the latency l_y, and $y \in \{1, \ldots, 4\}$ represents the possible labels, that are mapped to scenarios a) to d), respectively. We assumed a tolerance of 20 ms, and adopted as the correct label y, the scenario for which

$$\ell_y \leq \ell_{\max} + 20$$

and

$$y = \arg \min_{i=1,\ldots,4} (\ell_{\max} + 20 - \ell_i).$$

For instance, given that the SLA of a new slice specifies a maximum latency $\ell_{\max} = 100$ ms, we choose $y = 2$ as the correct label, which corresponds to scenario b), given that $\ell_2 = 115.4$ meets the requirement within the tolerance.

In spite of the fact that the correct label depends only on $\ell_{\max}$, we incorporated other eight parameters to create a DNN input vector with nine elements to illustrate the testbed flexibility. More details can be obtained via the provided source code, but in summary the sources of information to compose the DNN input vector are: the Ryu controller (that provided throughput, delay, and jitters in the fronthaul and backhaul), FlexRAN (e. g., number of deployed eNBs) and the requested slice KPIs (maximum latency $\ell_{\max}$ and minimum throughput). Table 3 shows the number of neurons in each layer of the adopted DNN model.

After the DNN is trained with the designed dataset, a proof-of-concept is conducted as follows. We consider a single slice serving three UEs and periodically change its associated SLA. At each SLA change, the nine DNN input values are collected and the DNN output is calculated. The VNFs are then placed according to the DNN output. Fig. 11 illustrates four examples of this process, with the SLA changing at each 15 s, and assuming the values of $\ell_{\max}$ over time are [150, 120, 100, 20]. These target values are indicated by horizontal dashed lines in Fig. 11.

Each time the DNN indicates a new output, the corresponding scenario is automatically deployed in the infrastructure, generating the mobile network with the defined topology and the connected UEs. The time taken by this process depends on several implementation details and is omitted in Fig. 11. During the first 15 seconds in Fig. 11, the SLA specifies $\ell_{\max} = 150$ ms and the DNN outputs $y = 1$, which corresponds to scenario a). The latency perceived by the three

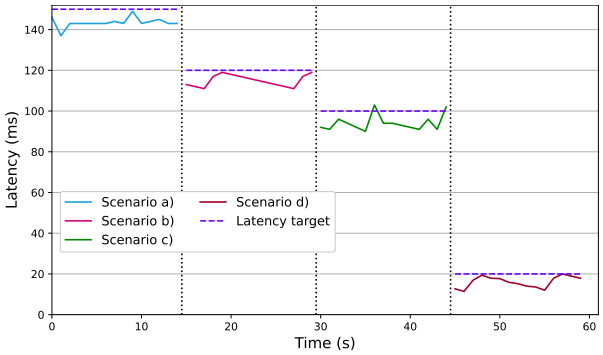

FIGURE 11. Average latency for distinct NFVs scenarios chosen by a DNN, adapting to a target latency (dashed lines) that changes every 15 seconds.

UEs are collected and their average stays below the target $\ell_{\max} = 150$ ms. Fig. 11 also shows that when the target latency is made more strict, the CAI testbed automatically adapts the NFVs towards scenario d), to successfully meet the SLA requirement. For instance, from 30 to 45 s, the SLA specifies $\ell_{\max} = 100$ ms and the DNN outputs $y = 3$, which corresponds to scenario c). In this interval, the impact of the tolerance of 20 ms can be perceived because the average latency is sometimes larger than $\ell_{\max}$.

Similar to the RAN slicing use case, we adopted DNN-based supervised learning for VNF allocation, but CAI can support other ML and AI paradigms such as RL, as well as take in account aspects such as energy consumption and network congestion.

V. OPEN ISSUES AND CONCLUSION

This paper presented a flexible 5G testbed with RAN and SDN controllers, connected via virtualized backhaul and fronthaul. The main focus of the proposed CAI testbed is enabling ML-based applications. When compared to other testbeds, CAI has the advantages of being reproducible among distinct sites given the provided containers, and lower cost than previously published alternatives.

For instance, strategically, CAI does not incorporate MANO components because they currently demand a relatively high-cost infrastructure. For instance, as indicated in [45] the ONAP-C MANO implementation, required 88 virtual CPUs and 176 GB of RAM memory. Some functionality of modern 5G / B5G networks are missing when not incorporating MANO, but many AI-based applications can still be implemented and assessed. To illustrate the flexibility of the proposed testbed, this paper presented two ML applications using CAI. They indicate how the AI agents can be integrated and assessed with respect to optimizing the instantiated mobile network.

The authors are now investigating the Elasticsearch stack to provide the storage of the controller's information in a time series format, improving real-time training of AI agents. Elasticsearch also facilitates gathering information from the computers used in the testbed, such as load, CPU and RAM

memory usage. We are also collecting large amount of data in order to conduct realistic experiments with unsupervised learning for anomaly detection and reinforcement learning for scheduling radio resources.

ACKNOWLEDGMENT

P. Batista, S. Lins and N. Linder contributions are theoretical concepts and they are with Ericsson Research.

REFERENCES

[1] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2429–2453, 3rd Quart., 2018.

[2] A. Imran and A. Zoha, "Challenges in 5G: How to empower SON with big data for enabling 5G," *IEEE Netw.*, vol. 28, no. 6, pp. 27–33, Nov. 2014.

[3] C. Benzaid and T. Taleb, "AI-driven zero touch network and service management in 5G and beyond: Challenges and research directions," *IEEE Netw.*, vol. 34, no. 2, pp. 186–194, Mar. 2020.

[4] M. E. Morocho-Cayamcela, H. Lee, and W. Lim, "Machine learning for 5G/B5G mobile and wireless communications: Potential, limitations, and future directions," *IEEE Access*, vol. 7, pp. 137184–137206, 2019.

[5] K. B. Letaief, W. Chen, Y. Shi, J. Zhang, and Y.-J.-A. Zhang, "The roadmap to 6G: AI empowered wireless networks," *IEEE Commun. Mag.*, vol. 57, no. 8, pp. 84–90, Aug. 2019.

[6] F. Wilhelmi, S. Barrachina-Muñoz, B. Bellalta, C. Cano, A. Jonsson, and V. Ram, "A flexible machine learning-aware architecture for future WLANs," 2019, *arXiv:1910.03510*. [Online]. Available: http://arxiv.org/abs/1910.03510

[7] K. B. Letaief, W. Chen, Y. Shi, J. Zhang, and Y.-J.-A. Zhang, "The roadmap to 6G: AI empowered wireless networks," *IEEE Commun. Mag.*, vol. 57, no. 8, pp. 84–90, Aug. 2019.

[8] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. London, U.K.: Pearson, Jul. 2016.

[9] *Architectural Framework for Machine Learning in Future Networks Including IMT-2020*, document ITU-T Rec. Y.3172, 2019.

[10] *Study of Enablers for Network Automation for 5G*, document TR 23.791, 3GPP, 2019.

[11] *Draft Zero-Touch Network and Service Management (ZSM): Reference Architecture*, Standard ETSI GS ZSM 002 V0.13.5 (2019-07), 2019.

[12] L. Deng, H. Deng, and A. Mayer. *Harmonizing Open Source and Standards: A Case for 5G Slicing*. Accessed: Sep. 30, 2020. [Online]. Available: https://www.onap.org/wp-content/uploads/sites/20/2020/03/ONAP_HarmonizingOpenSourceStandards_031520.pdf

[13] K. Tan, D. Wu, A. Chan, and P. Mohapatra, "Comparing simulation tools and experimental testbeds for wireless mesh networks," *Pervas. Mobile Comput.*, vol. 7, no. 4, pp. 434–448, Aug. 2011.

[14] A. Rostami, P. Ohlen, K. Wang, Z. Ghebretensae, B. Skubic, M. Santos, and A. Vidal, "Orchestration of RAN and transport networks for 5G: An SDN approach," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 64–70, Apr. 2017.

[15] R. Munoz, L. Nadal, R. Casellas, M. S. Moreolo, R. Vilalta, J. M. Fabrega, R. Martinez, A. Mayoral, and F. J. Vilchez, "The ADRENALINE testbed: An SDN/NFV packet/optical transport network and edge/core cloud platform for end-to-end 5G and IoT services," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Jun. 2017, pp. 1–5.

[16] S. Fichera, M. Gharbaoui, P. Castoldi, B. Martini, and A. Manzalini, "On experimenting 5G: Testbed set-up for SDN orchestration across network cloud and IoT domains," in *Proc. IEEE Conf. Netw. Softwarization (NetSoft)*, Jul. 2017, pp. 1–6.

[17] K. Ramantas, A. Antonopoulos, E. Kartsakli, P.-V. Mekikis, J. Vardakas, and C. Verikoukis, "A C-RAN based 5G platform with a fully virtualized, SDN controlled optical/wireless fronthaul," in *Proc. 20th Int. Conf. Transparent Opt. Netw. (ICTON)*, Jul. 2018, pp. 1–4.

[18] A. Thantharate, R. Paropkari, V. Walunj, and C. Beard, "DeepSlice: A deep learning approach towards an efficient and reliable network slicing in 5G networks," in *Proc. IEEE 10th Annu. Ubiquitous Comput., Electron. Mobile Commun. Conf. (UEMCON)*, Oct. 2019, pp. 0762–0767.

[19] G. Zhu, J. Zan, Y. Yang, and X. Qi, "A supervised learning based QoS assurance architecture for 5G networks," *IEEE Access*, vol. 7, pp. 43598–43606, 2019.

[20] P. Du and A. Nakao, "Deep learning-based application specific RAN slicing for mobile networks," in *Proc. IEEE 7th Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2018, pp. 1–3.

[21] N. Nikaein, C.-Y. Chang, and K. Alexandris, "Mosaic5G: Agile and flexible service platforms for 5G research," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 48, no. 3, pp. 29–34, Sep. 2018.

[22] G. Garcia-Aviles, M. Gramaglia, P. Serrano, and A. Banchs, "POSENS: A practical open source solution for end-to-end network slicing," *IEEE Wireless Commun.*, vol. 25, no. 5, pp. 30–37, Oct. 2018.

[23] C.-Y. Huang, C.-Y. Ho, N. Nikaein, and R.-G. Cheng, "Design and prototype of a virtualized 5G infrastructure supporting network slicing," in *Proc. IEEE 23rd Int. Conf. Digit. Signal Process. (DSP)*, Nov. 2018, pp. 1–5.

[24] G. M. Yilma, F. Zarrar Yousaf, V. Sciancalepore, and X. Costa-Perez, "On the challenges and KPIs for benchmarking open-source NFV MANO systems: OSM vs ONAP," 2019, *arXiv:1904.10697*. [Online]. Available: http://arxiv.org/abs/1904.10697

[25] Z. Li, M. Kihl, Q. Lu, and J. A. Andersson, "Performance overhead comparison between hypervisor and container based virtualization," in *Proc. IEEE 31st Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, Mar. 2017, pp. 955–962.

[26] E. A. Brewer, "Kubernetes and the path to cloud native," in *Proc. 6th ACM Symp. Cloud Comput. (SoCC)*, 2015, p. 167.

[27] *Secure Networking for the Cloud Native Era*. Accessed: Sep. 13, 2020. [Online]. Available: https://www.projectcalico.org/

[28] *OpenAirInterface—5G Software Alliance for Democratising Wireless Innovation*. Accessed: Sep. 13, 2020. [Online]. Available: http://www.openairinterface.org/

[29] A. Checko, H. L. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. S. Berger, and L. Dittmann, "Cloud RAN for mobile networks—A technology overview," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 1, pp. 405–426, 1st Quart., 2014.

[30] B. B. Rad, H. J. Bhatti, and M. Ahmadi, "An introduction to Docker and analysis of its performance," *Int. J. Comput. Sci. Netw. Secur.*, vol. 17, no. 3, p. 228, 2017.

[31] Free5GC. (2019). *Free5GC: Open-Source 5GC*. [Online]. Available: https://www.free5gc.org/

[32] Mininet. *Mininet: An Instant Virtual Network on your Laptop (or other PC)*. Accessed: Sep. 16, 2020. [Online]. Available: http://mininet.org/

[33] S. Asadollahi, B. Goswami, and M. Sameer, "Ryu controller's scalability experiment on software defined networks," in *Proc. IEEE Int. Conf. Current Trends Adv. Comput. (ICCTAC)*, Feb. 2018, pp. 1–5.

[34] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis, "FlexRAN: A flexible and programmable platform for software-defined radio access networks," in *Proc. 12th Int. Conf. Emerg. Netw. EXperiments Technol.*, Dec. 2016, pp. 427–441.

[35] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.

[36] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.

[37] E. Bisong, "Kubeflow and kubeflow pipelines," in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Berkeley, CA, USA: Springer, 2019, pp. 671–685.

[38] P. Du and A. Nakao, "Deep learning-based application specific RAN slicing for mobile networks," in *Proc. IEEE 7th Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2018, pp. 1–3.

[39] S. Costanzo, I. Fajjari, N. Aitsaadi, and R. Langar, "A network slicing prototype for a flexible cloud radio access network," in *Proc. 15th IEEE Annu. Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2018, pp. 1–4.

[40] A. Ksentini and N. Nikaein, "Toward enforcing network slicing on RAN: Flexibility and resources abstraction," *IEEE Commun. Mag.*, vol. 55, no. 6, pp. 102–108, 2017.

[41] B. Khodapanah, A. Awada, I. Viering, A. N. Barreto, M. Simsek, and G. Fettweis, "Slice management in radio access network via iterative adaptation," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–7.

[42] R. Schmidt, C.-Y. Chang, and N. Nikaein, "Slice scheduling with QoS-guarantee towards 5G," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2019, pp. 1–7.

[43] F. Chollet, *Deep Learning With Python*. Shelter Island, NY, USA: Manning Publications, 2018.

[44] F. D. Calabrese, L. Wang, E. Ghadimi, G. Peters, L. Hanzo, and P. Soldati, "Learning radio resource management in RANs: Framework, opportunities, and challenges," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 138–145, Sep. 2018.

[45] G. M. Yilma, Z. F. Yousaf, V. Sciancalepore, and X. Costa-Perez, "Benchmarking open source NFV MANO systems: OSM and ONAP," *Comput. Commun.*, vol. 161, pp. 86–98, Sep. 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366420305946

CLEVERSON VELOSO NAHUM was born in Brazil. He received the B.Sc. degree in computer engineering from the Federal University of Pará (UFPA), Belém, Pará, Brazil, in 2019. He is currently pursuing the master's degree in electrical engineering with emphasis on telecommunications with the Electrical Engineering Graduate Program, UFPA. He is part of the Research and Development Center for Telecommunications, Automation and Electronics (LASSE), since 2016. His current research interests include network slicing, network functions virtualization, and artificial intelligence applied on mobile communication systems.

LUCAS DE NÓVOA MARTINS PINTO was born in Brazil. He is currently pursuing the degree in electrical engineering with the Federal University of Pará (UFPA), Belém, Pará, Brazil. He is also part of the Research and Development Center for Telecommunications, Automation and Electronics (LASSE). He has experience in the following subjects: 2G/4G/5G mobile networks, community networks, mobile networks, and transport layer.

VIRGÍNIA BRIOSO TAVARES was born in Brazil. She is currently pursuing the degree in electrical engineering with the Federal University of Pará. She is part of the Research and Development Center for Telecommunications, Automation and Electronics (LASSE), since 2018. She has experience in the area of electrical engineering, with emphasis on telecommunications, acting mainly in the following subjects: digital communications, community networks, 2G/ 4G/ 5G mobile networks, and network virtualization.

PEDRO BATISTA received the B.S., M.S., and Ph.D. degrees from the Electrical Engineering Graduate Program, Federal University of Pará, Brazil. He is currently a Researcher with Ericsson. His research interests are in optimization of future mobile networks, particularly, using machine learning and machine reasoning, and future internet architectures.

SILVIA LINS received the B.Sc. degree in computer engineering and the M.Sc. degree in electrical engineering from the Federal University of Pará (UFPA), Brazil, where she is currently pursuing the Ph.D. degree in telecommunications. She is currently an Experienced Researcher in the network architecture and protocols area with Ericsson Brazil. Her current research involves machine learning for 5G and beyond network scenarios and applications. In the past, she also worked with information-centric networks, transport networks, and 3G/4G mobile networks.

NEIVA LINDER received the Ph.D. degree in electrical engineering with major in telecommunication from the Federal University of Pará (UFPA), Belém, Brazil. She is currently the Research Manager with the Network Management and Automation team, Research Area Networks, Ericsson Research, Sweden. She has research interests on AI based operations applied to mobile networks automation and service assurance. She joined Ericsson, in 2011, and has worked in several technology areas applied to fixed and mobile backhaul network architectures, transport solutions for 4G/5G, including Cloud RAN, SDN, NFV, and network slicing. She has over ten years of experience in telecommunication. Previously to Ericsson, she was active in the area of signal processing for communication and held a postdoctoral position at the EIT – LTH Faculty of Engineering, Lund University, Sweden.

ALDEBARO KLAUTAU (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of California at San Diego (UCSD), in 2003. He is currently a Full Professor with the Federal University of Pará (UFPA), where he is the ITU-T Focal Point and the coordinator of LASSE. He is a Researcher with CNPq, Brazil. His work focuses on machine learning and signal processing for telecommunications.