This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3342711

OrchestRAN: Orchestrating Network Intelligence in the Open RAN

Salvatore D'Oro, *Member, IEEE*, Leonardo Bonati, *Member, IEEE*, Michele Polese, *Member, IEEE*, and Tommaso Melodia, *Fellow, IEEE*

Abstract—The next generation of cellular networks will be characterized by softwarized, open, and disaggregated architectures exposing analytics and control knobs to enable network intelligence via innovative data-driven algorithms. How to practically realize this vision, however, is largely an open problem. Specifically, for a given intent, it is still unclear how to select which data-driven models should be deployed and where, which parameters to control, and how to feed them appropriate inputs. In this paper, we take a decisive step forward by presenting OrchestRAN, a network intelligence orchestration framework for next generation systems that embraces and builds upon the Open Radio Access Network (RAN) paradigm to provide a practical solution to these challenges. OrchestRAN has been designed to execute in the non-Real-time (RT) RAN Intelligent Controller (RIC) as an rApp and allows Network Operators (NOs) to specify high-level control/inference objectives (i.e., adapt scheduling, and forecast capacity in near-RT, e.g., for a set of base stations in Downtown New York). OrchestRAN automatically computes the optimal set of data-driven algorithms and their execution location (e.g., in the cloud, or at the edge) to achieve intents specified by the NOs while meeting the desired timing requirements and avoiding conflicts between different data-driven algorithms controlling the same parameters set. We show that the intelligence orchestration problem in Open RAN is NP-hard. To support real-world applications, we also propose three complexity reduction techniques to obtain low-complexity solutions that, when combined, can compute a solution in 0.1 s for large network instances. We prototype OrchestRAN and test it at scale on Colosseum, the world's largest wireless network emulator with hardware in the loop. Our experimental results on a network with 7 base stations and 42 users demonstrate that OrchestRAN is able to instantiate data-driven services on demand with minimal control overhead and latency.

Index Terms—O-RAN, Open RAN, Artificial Intelligence, Orchestration, 5G, 6G.

1 Introduction

The fifth-generation (5G) of cellular networks and its evolution (NextG), will mark the end of the era of inflexible hardware-based Radio Access Network (RAN) architectures in favor of innovative and agile solutions built upon softwarization, openness and disaggregation principles. This paradigm shift—often referred to as Open RAN—comes with unprecedented flexibility. It makes it possible to split

The authors are with the Institute for the Wireless Internet of Things, Northeastern University, Boston, MA, USA. Email: {s.doro, l.bonati, m.polese, melodia}@northeastern.edu.

This work was partially supported by the U.S. National Science Foundation under Grants CNS-1923789 and NSF CNS-1925601, and the U.S. Office of Naval Research under Grant N00014-20-1-2132.

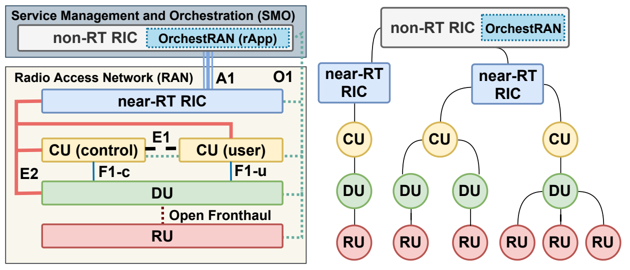

Figure 1. O-RAN reference architecture and interfaces (left). Representation of an O-RAN network architecture as a tree graph (right).

network functionalities—traditionally embedded and executed in monolithic base stations—and instantiate and control them across multiple nodes of the network [1].

1.1 The O-RAN Paradigm

In this context, the O-RAN Alliance [2], a consortium led by Network Operators (NOs), vendors and academic partners, is developing a standardized architecture for Open RAN that promotes horizontal disaggregation and standardization of RAN interfaces, thus enabling multivendor equipment interoperability and algorithmic network control and analytics [3]. As shown in Fig. 1, O-RAN embraces the 3rd Generation Partnership Project (3GPP) functional split with Central Units (CUs), Distributed Units (DUs) and Radio Units (RUs) implementing different functions of the protocol stack. O-RAN also introduces (i) a set of open standardized interfaces to interact, control and collect data from every node of the network; as well as (ii) RAN Intelligent Controllers (RICs) that execute third-party applications over an abstract overlay to control RAN functionalities, i.e., *xApps* in the near-Real-time (RT) and *rApps* in the non-RT RIC.

The O-RAN architecture makes it possible to bring automation and intelligence to the network through Machine Learning (ML) and Artificial Intelligence (AI), which will leverage the enormous amount of data generated by the RAN—and exposed through the O-RAN interfaces—to analyze the current network conditions, forecast future traffic profiles and demand, and implement closed-loop network control strategies to optimize the RAN performance.

Authorized licensed use limited to: Northeastern University. Downloaded on January 29,2024 at 14:47:29 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3342711

1.2 The Importance of AI in O-RAN

For this reason, how to design, train and deploy reliable and effective data-driven solutions has recently received increasing interest from academia and industry alike, with applications ranging from controlling RAN resource and transmission policies [4–15], to forecasting and classifying traffic and Key Performance Indicators (KPIs) [16–20], as well as regulating the placement of network components and functions [21, 22], thus highlighting how these approaches will be foundational to the Open RAN paradigm. However, how to deploy and manage, i.e., *orchestrate*, intelligence into softwarized cellular networks is a uniquely challenging problem for the following reasons:

• *Complying with time scales and making input available:* Adapting RAN parameters and functionalities requires control loops operating over time scales ranging from a few milliseconds (i.e., real-time) to a few hundreds of milliseconds (i.e., near-RT) to several seconds (i.e., non-RT) [8, 23]. As a consequence, the models and the location where they are executed need to be selected to be able to retrieve the necessary inputs and compute the output within the appropriate time constraints [8, 24]. For instance, while IQ samples are easily available in real time at the RAN, it is extremely hard to make them available at the near-RT and non-RT RICs within the same temporal window, making the execution of models that require IQ samples as input on the RICs ineffective [25].

• *Choosing the right model:* Each ML/AI model is designed to accomplish specific inference and/or control tasks and requires well-defined inputs in terms of data type and size. One must make sure that the most suitable model is selected for a specific NO request, and that it meets the required performance metrics (e.g., minimum accuracy), delivers the desired inference/control functionalities, and is instantiated on nodes with enough resources to execute it.

• *Conflict mitigation:* One must also ensure that selected ML/AI models do not conflict with each other, and that the same parameter (or functionality) is controlled by only a single model at any given time.

We also mention that, despite similar in certain aspects, this problem cannot be trivially solved using existing approaches for Virtual Network Function (VNF), services [26] and virtual machine placement [27]. Indeed, solutions for cloud-native architectures primarily focuses on orchestrating functionalities that deliver services such as computer vision, video transcoding, firewalls and content delivery (to name a few) and very rarely deal with functions that control other network elements (e.g., RAN components) which comes with unique challenges such as: (i) ensuring that each ML/AI model receives the proper input and within the desired temporal windows so as to process data that is not obsolete and ensure timely decision making; (ii) selecting the correct ML/AI model that meets NO intents in terms of accuracy and KPIs; and (iii) avoid conflicts that might result in inefficient decision making and performance degradation.

For these reasons, *orchestrating* network intelligence in the Open RAN presents unprecedented and unique challenges that call for innovative, automated and scalable solutions. In this paper, we address these challenges by presenting OrchestRAN, an automated intelligence orchestration framework for the Open RAN. OrchestRAN follows the O-RAN specifications and operates as an rApp executed in the non-RT RIC (Fig. 1) providing automated routines to: (i) Collect requests from NOs; (ii) select the optimal ML/AI models to achieve NOs' goals and avoid conflicts; (iii) determine the optimal execution location for each model complying with timescale requirements, resource and data availability, and (iv) automatically embed models into O-RAN applications that are dispatched to selected nodes, where they are executed and fed the required inputs.

1.3 Main Contributions

To achieve this goal, we have designed and prototyped novel algorithms embedding pre-processing variable reduction and branching techniques that allow OrchestRAN to compute orchestration solutions with different complexity and optimality trade-offs, while ensuring that the NOs intents are satisfied. We evaluate the performance of OrchestRAN in orchestrating intelligence in the RAN through numerical simulations, and by prototyping OrchestRAN on ColO-RAN [28], an O-RAN-compliant large-scale experimental platform developed on top of Colosseum, the world's largest wireless network emulator with hardware in-the-loop [29]. Experimental results on an O-RAN-compliant softwarized network with 7 cellular base stations and 42 users demonstrate that OrchestRAN enables seamless instantiation of O-RAN applications with diverse timescale requirements at different O-RAN components. OrchestRAN automatically selects the optimal execution locations for each O-RAN application, thus moving network intelligence to the edge with up to 2.6× reduction of control overhead over the O-RAN E2 interface. To the best of our knowledge, this is the first large-scale demonstration of an O-RAN-compliant network intelligence orchestration system.

This manuscript extends the shorter conference paper version [30] by providing (i) extensive details on the prototype implementation of OrchestRAN, (ii) an in-depth discussion on the advantages of ML/AI model sharing, (iii) details on the design and training of the ML/AI solutions used in the prototype; and (iv) more numerical and experimental results to assess the performance of our solution.

The remainder of this paper is organized as follows: Section 2 surveys related work, while Section 3 introduces the O-RAN architecture. OrchestRAN, its building blocks and procedures are presented in Section 4. Section 5 formulates the network intelligence orchestration problem and shows its NP-hardness, while Section 6 presents low-complexity and scalable algorithms to solve it. OrchestRAN performance are assessed numerically in Section 7, while Sections 8 and 9 present the OrchestRAN prototype and its performance evaluation, respectively. Finally, Section 10 concludes the paper.

2 Related Work

The application of ML/AI algorithms to cellular networks is gaining momentum as a promising and effective way

Authorized licensed use limited to: Northeastern University. Downloaded on January 29,2024 at 14:47:29 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3342711

to design and deploy solutions capable of predicting, controlling, and automating the network behavior under dynamic conditions. Relevant examples include the application of Deep Learning and Deep Reinforcement Learning (DRL) to predict the network load [10, 16, 31], classify traffic [17, 32, 33], perform beam alignment [18, 19], allocate radio resources [4, 5, 15, 34], and deploy service-tailored network slices [6–11, 35]. It is clear that ML/AI techniques will play a key role in the transition to intelligent networks, especially in the O-RAN ecosystem [36]. However, a relevant challenge that still remains unsolved is how to bring such intelligence to the network in an efficient, reliable and automated way, which is ultimately the goal of this paper.

In [37], Ayala-Romero et al. present an online Bayesian learning orchestration framework for intelligent virtualized RANs where resource allocation follow channel conditions and network load. The same authors present a similar framework in [12], where networking and computational resources are orchestrated via DRL to comply with service level agreements (SLAs) while accounting for the limited amount of resources. Singh et al. present GreenRAN, an energy-efficient orchestration framework for NextG that splits and allocates RAN components according to the current resource availability [38]. In [39], Chatterjee et al. present a radio resource orchestration framework for 5G applications where network slices are dynamically re-assigned to avoid inefficiencies and SLA violations. Puligheddu et al. design a semantic framework compliant with the O-RAN specifications to allocate computational resources at the edge based on the complexity of the tasks to perform, with a focus on image recognition [40]. Relevant to our work are the works of Morais et al. [22] and Matoussi et al. [41], which present frameworks to optimally disaggregate, place and orchestrate RAN components in the network to minimize computation and energy consumption while accounting for diverse latency and performance requirements. Although the above works all present orchestration frameworks for NextG systems, they are focused on orchestrating RAN resources and functionalities, rather than network intelligence, which represents a substantially different problem.

In the context of orchestrating ML/AI models in NextG systems, Baranda et al. [42, 43] present an architecture for the automated deployment of models in the 5Growth management and orchestration (MANO) platform [44], and demonstrate automated instantiation of models on demand. The closest to our work is the work of Salem et al. [24], which proposes an orchestrator to select and instantiate inference models at different locations of the network to obtain a desirable balance between accuracy and latency. However, [24] is not concerned with O-RAN systems, but focuses on data-driven solutions for inference in cloud-based applications.

Besides the differences highlighted in the previous discussion, OrchestRAN differs from the above works in that it focuses on the Open RAN architecture and is designed to instantiate *both* inference and control solutions complying with O-RAN specifications. Moreover, OrchestRAN allows model sharing across multiple requests to efficiently reuse available network resources. We prototyped and benchmarked OrchestRAN on Colosseum. To the best of our knowledge, this is the first large-scale demonstration of a network intelligence orchestration system tailored to O-RAN architecture and networks.

3 O-RAN: A Primer

O-RAN embraces the 7-2x functional split (an extension of the 3GPP 7-2 split), where network functionalities are divided across multiple nodes, namely, CUs, DUs and RUs (Fig. 1, left). The RUs implement lower physical layer functionalities. The DUs interact with the RUs via the Open Fronthaul interface [45] and implement functionalities pertaining to both the higher physical layer and the MAC layer. Finally, the remaining functionalities of the protocol stack are implemented and executed in the CU. The latter is connected to the DUs through the F1 interface and is further split in two entities—handling control and user planes—connected via the E1 interface. These network elements run on “white-box” hardware components connected through O-RAN open interfaces, thus enabling multivendor interoperability and overcoming the vendor lock-in [1].

Beyond disaggregation, the main innovation introduced by O-RAN lies in the non-RT and near-RT RICs. These components enable dynamic and softwarized control of the RAN, as well as the collection of statistics via a publish-subscribe model [46] through open and standardized interfaces, e.g., the O1 and E2 interfaces (Fig. 1, left). Specifically, the near-RT RIC hosts applications (*xApps*) that implement time-sensitive—i.e., between 10 ms and 1 s—operations to perform closed-loop control over the RAN elements. Practical examples include control of load balancing, handover procedures, scheduling and RAN slicing policies [8, 47–49]. The non-RT RIC, instead, is designed to execute within a service management and orchestration (SMO) framework, e.g., Open Network Automation Platform (ONAP), and acts at time scales above 1 s. It takes care of training ML/AI models, as well as deploying models and network control policies on the near-RT RIC through the A1 interface. Similar to its near-RT counterpart, the non-RT RIC supports the execution of third-party applications, called *rApps*. These components act in concert to gather data and performance metrics from the RAN, and to optimize and reprogram its behavior in real time through software algorithms to reach NO’s goals.

Since O-RAN embraces virtualization and cloud-native principles, all of the above components can be instantiated and executed both at the cell site or in the cloud, except for RUs which must be deployed on the cell site as they host the antennas necessary to establish wireless links. In this paper, we consider the O-RAN deployment scenario B [1], but OrchestRAN also applies to any other deployment scenarios.

O-RAN specifications also envision ML/AI models instantiated directly on the CUs and DUs, implementing RT—Transmission Time Interval (TTI) level—control loops that operate on 10 ms time-scales [50]. As of today, these real-time control loops have not been yet specified by the O-RAN Alliance, but *dApps* [25] represent a preliminary effort to extend O-RAN specifications and their ML/AI capabilities to CUs/DUs in order to support applications for 6G and beyond. To ensure the highest level of flexibility and extendablity, OrchestRAN has been natively designed to

Authorized licensed use limited to: Northeastern University. Downloaded on January 29,2024 at 14:47:29 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3342711

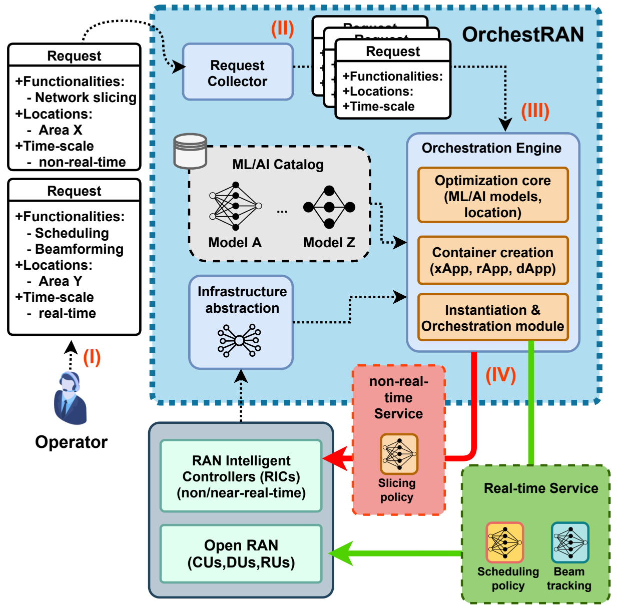

Figure 2. System design of OrchestRAN and main procedures.

support control loops implementing RT applications, such as dApps [25].

4 OrchestRAN

As illustrated in Fig. 1, OrchestRAN is designed to be executed as an *rApp* at the non-RT RIC. Its architecture is illustrated in Fig. 2. It consists of: (i) The *Infrastructure Abstraction* module, abstracting relevant information on the underlying physical infrastructure (e.g., available resources, deployment location of RICs and RAN components); (ii) the *ML/AI Catalog*, storing data-driven solutions that can be instantiated in the network, (iii) the *Request Collector*, collecting the requests from the NOs, and (iv) the *Orchestration Engine*, orchestrating and instantiating O-RAN applications to satisfy the NOs requests.

At a high-level, first NOs specify their intent by submitting a request to OrchestRAN (step I). This includes the set of functionalities they want to deploy (e.g., network slicing, beamforming, scheduling control, etc.), the location where functionalities are to be executed (e.g., RIC, CU, DU) and the desired time constraint (e.g., delay-tolerant, low-latency). Then, requests are gathered by the *Request Collector* (step II, Section 4.3) and fed to the *Orchestration Engine* (step III, Section 4.4) which: (i) Accesses the *ML/AI Catalog* (Section 4.2) and the *Infrastructure Abstraction* module (Section 4.1) to determine the optimal orchestration policy and models to be instantiated; (ii) automatically creates containers with the embedded ML/AI models in the form of O-RAN applications, and (iii) dispatches such applications at the locations determined by the Orchestration Engine.

Since NOs goals and intents are likely to vary over time, the above procedures are executed periodically on a time-slotted basis.

For the sake of clarity, in Table 1 we have summarized the notation used in the next sections.

4.1 The Infrastructure Abstraction Module

This module provides a high-level representation of the physical RAN architecture. Following the O-RAN specifications, this is divided into five separate logical groups as shown in Fig. 1 (right): non-RT RICs, near-RT RICs, CUs, DUs and RUs.

Each group contains a different number of nodes deployed at different locations of the network. Let $\mathcal{D}$ be the set of such nodes, and $D = |\mathcal{D}|$ be their number.

As discussed before, the 7-2x functional split leverages a hierarchical architecture where, for example, the near-RT RIC controls multiple CUs/DUs, and the non-RT RIC oversees multiple near-RT RICs. These hierarchical relationships can be represented via an undirected graph with a tree structure such as the one in Fig. 1 (right). Specifically, leaves represent nodes at the edge (e.g., RUs/DUs/CUs), while the non-RT RIC is the root of the tree. It is also worth noticing that CUs/DUs/RUs might be coexisting on the same node. To capture this property, coexisting CUs/DUs/RUs are modeled as a single logical node with a hierarchy level equal to that of the hierarchically highest node in the group (e.g., co-located DU and RU can be represented as a leaf with the hierarchy level of a DU).

For any two nodes $d', d'' \in \mathcal{D}$, we define variable $c_{d',d''} \in \{0,1\}$ such that $c_{d',d''} = 1$ if node d' is *reachable* from node d'', $c_{d',d''} = 0$ otherwise. It is worth mentioning that "reachability" is not tied to "graph connectivity". Specifically, while two nodes d' and d'' might be connected in the general sense of graphs (as shown in Fig. 1 (right) where leaves from a branch are connected to leaves of another branch), they might not be physically reachable in the networking sense. For example, while all RUs belong to the same infrastructure graph tree (i.e., they belong to a connected graph), they are likely unreachable from one another due to either the lack of physical connectivity links, limited network visibility, existence of firewalls, or due to the high latency required to transfer data from one node to another.

Furthermore, for each node d, let ρ_d^ξ be the total amount of resources of type $\xi \in \Xi$ dedicated to hosting and executing ML/AI models and their functionalities, where Ξ represents the set of all resource types. Although we do not make any assumptions on the specific types of resources, practical examples may include the number of CPUs, GPUs, as well as available disk storage and memory. In the following, we assume that each non-RT RIC identifies an independent networking domain and the set of nodes $\mathcal{D}$ includes near-RT RICs, CUs, DUs and RUs controlled by the corresponding non-RT RIC only.

4.2 The ML/AI Catalog

In OrchestRAN, the available pre-trained data-driven solutions are stored in a *ML/AI Catalog* consisting of a set $\mathcal{M}$ of ML/AI models. Although our framework is general and supports both offline and online data-driven solutions, it is worth mentioning that O-RAN specifications dictate that all models contained in the ML/AI Catalog must have been already pre-trained offline in the non-RT RIC [50], and have successfully passed validation tests to ensure their effectiveness and reliability once deployed on the field.

Authorized licensed use limited to: Northeastern University. Downloaded on January 29,2024 at 14:47:29 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3342711

Table 1
Summary of Notation

Variable	Description
$\mathcal{D}$, $D = \lvert\mathcal{D}\rvert$	Set of nodes in the network and their number
$\mathcal{F}$, $F = \lvert\mathcal{F}\rvert$	Set of control and inference functionalities and their number
$\mathcal{M}$, $M = \lvert\mathcal{M}\rvert$	Set of ML/AI models in the catalog and their number
$\mathcal{F}_m \subseteq \mathcal{F}$	Set of functionalities offered by model $m \in \mathcal{M}$
$\mathcal{I}$, $I = \lvert\mathcal{I}\rvert$	Set of outstanding requests and their number
$\mathcal{T}$	Set of ML/AI model input types
$c_{d,d'}$	Binary variable to indicate if node d can reach node d'
Ξ	Set of resource types
ρ_d^ξ	Amount of resources of type $\xi \in \Xi$ available at node d
ρ_m^ξ	Amount of resources of type $\xi \in \Xi$ required by model m
$\sigma_{m,f}$	Binary variable to indicate if model m offers functionality f
$t_m^{\mathrm{IN}} \in \mathcal{T}$	Input type required by model m
$\beta_{m,f,d}$	Suitability indicator for model m to deliver f at node d
$\gamma_{m,f}$	Performance score for model m to deliver f
$C_{m,d}$	Maximum number of instances of model m on node d
$\mathcal{F}_{i,d}$	Set of functionalities demanded by request i on node d
$\mathcal{F}_i$	Set of functionalities demanded by request $i \in \mathcal{I}$ on all nodes
v_i	Monetary value of request i
$\tau_{i,f,d}$	Binary variable to indicate if $f \in \mathcal{F}_{i,d}$
$\mathcal{D}_i$	Set of nodes where functionalities in $\mathcal{F}_i$ should be offered
$\pi_{i,f,d}$	Minimum performance requirement to execute functionality f on node d and satisfy request i
$\boldsymbol{\pi}_i$	Set of all performance requirements for request i
$\delta_{i,f,d}$	Maximum tolerable latency to execute functionality f on node d and satisfy request i
$\mathcal{D}_{i,f,d}^{\mathrm{IN}}$	Set of nodes providing input to function f to be executed on node d as requested by i
$\mathcal{D}_i^{\mathrm{IN}}$	Set of input requirements for request i
$x_{m,k,d'}^{i,f,d}$	Binary optimization variable to indicate if function f requested by i on node d is execute via instance k of model m on node d'
y_i	Binary optimization variable to indicate if request i is being satisfied
$z_{m,k,d}$	Binary optimization variable to indicate if instance k of model m is being actively used on node d
$\mathbf{x}$	Orchestration policy
$\Delta_{i,f,d}$	Data collection time
$s_{t_m^{\mathrm{IN}}}$	The input size of model m measured in bytes
$b_{d,d'}$	Bottleneck data rate over the link between nodes d and d'
$T_{d,d'}$	Propagation delay between nodes d and d'
T_m^{exec}	Inference time for model m
$\Delta_{i,f,d}^{EXEC}$	Execution time

Let $\mathcal{F}$ be the set of all possible control and inference *functionalities* (e.g., scheduling, beamforming, capacity forecasting, handover prediction) offered by such ML/AI models—hereafter referred to simply as "models". This set includes all of the functionalities currently offered by all dApps, xApps and rApps in the ML/AI catalog. New functionalities are added to $\mathcal{F}$ whenever a new application is published in the catalog.

Let $M = |\mathcal{M}|$ and $F = |\mathcal{F}|$. For each model $m \in \mathcal{M}$, $\mathcal{F}_m \subseteq \mathcal{F}$ represents the subset of functionalities offered by m. Accordingly, we define a binary variable $\sigma_{m,f} \in \{0,1\}$ such that $\sigma_{m,f} = 1$ if $f \in \mathcal{F}_m$, $\sigma_{m,f} = 0$ otherwise. Clearly, we do not exclude the case where two (or more) models m' and m'' provide the same set of functionalities, i.e., $\mathcal{F}_{m'} = \mathcal{F}_{m''}$, but differ in terms of model size, architecture as well as required inputs.

To capture those characteristics, we use ρ_m^ξ to indicate the amount of resources of type ξ required to instantiate and execute model m. Moreover, we introduce $\mathcal{T}$ as the set of possible input types. For each model m, $t_m^{\mathrm{IN}} \in \mathcal{T}$ represents the type of input required by the model (e.g., IQ samples, throughput and buffer size measurements). The procedures regulating data gathering, the format and the periodicity at which data is collected from RAN nodes are specified by O-RAN specifications and an overview can be found in [3].

Naturally, not all models can be equally executed everywhere. For example, a model m performing beam alignment [18], in which received IQ samples are fed to a neural network to determine the beam direction, can only execute on nodes where IQ samples are available. While IQ samples can be accessed in real-time at the RU, it is hard to make them available at CUs and the RICs without incurring in high overhead and transmission latency with cases where a 100 *Gbps* link dedicated to streaming IQ samples only is needed to maintain a latency below 100 *ms* [25].

For this reason, we introduce a suitability indicator $\beta_{m,f,d} \in [0,1]$ which specifies how well a model m is suited to provide a specific functionality f when instantiated on node d. Values of $\beta_{m,f,d}$ closer to 1 mean that the model is well-suited to execute at a specific location, while values closer to 0 indicate that the model performs poorly, for example, due to the lack of input data at node d, or due to high computation latency that violates timing requirements (e.g., real-time) of that functionality. We also introduce a performance score $\gamma_{m,f}$ measuring the performance of the model with respect to f. Typical performance metrics include classification/forecasting accuracy, mean squared error and probability of false alarm.

A model can be instantiated on the same node multiple times to serve different NOs or traffic classes. However, due to limited resources, each node d supports at most $C_{m,d} = \min_{\xi\in\Xi}\{\lfloor \rho_d^\xi / \rho_m^\xi \rfloor\}$ instances of model m, where $\lfloor\cdot\rfloor$ is the floor operator.

It is worth mentioning that AI/ML models, and the O-RAN applications that embed them, are published in the catalog only after they have been deemed reliable and safe to be deployed on a production network. This validation is a critical component of the AI/ML lifecycle management in O-RAN and a detailed overview of this process is described in [3].

4.3 Request Collector

OrchestRAN allows NOs to submit requests specifying which functionalities they require, where they should execute, and the desired performance and timing requirements. Without loss of generality, we assume that each request is feasible as any unfeasible request would be naturally be excluded from the optimization process executed within the Orchestration Engine. The Request Collector of OrchestRAN is in charge of collecting such requests. A request i is defined as a tuple $(\mathcal{F}_i, \boldsymbol{\pi}_i, \boldsymbol{\delta}_i, \mathcal{D}_i^{\mathrm{IN}})$, with each element defined as follows:

Authorized licensed use limited to: Northeastern University. Downloaded on January 29,2024 at 14:47:29 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3342711

• *Functions and locations.* For each request i, we define the set of functionalities that must be instantiated on the nodes as $\mathcal{F}_i = (\mathcal{F}_{i,d})_{d\in\mathcal{D}}$, with $\mathcal{F}_{i,d} \subseteq \mathcal{F}$. Required functionalities and nodes are specified by a binary indicator $\tau_{i,f,d} \in \{0,1\}$ such that $\tau_{i,f,d} = 1$ if request i requires functionality f on node d, i.e., $f \in \mathcal{F}_{i,d}$, $\tau_{i,f,d} = 0$ otherwise. We also define $\mathcal{D}_i = \{d \in \mathcal{D} : \sum_{f\in\mathcal{F}_i} \tau_{i,f,d} \geq 1\}$ as the subset of nodes of the network where functionalities in $\mathcal{F}_i$ should be offered;

• *Performance requirements.* For any request i, $\boldsymbol{\pi}_i = (\pi_{i,f,d})_{d\in\mathcal{D}_i, f\in\mathcal{F}_{i,d}}$ indicates the minimum performance requirements that must be satisfied to accommodate i. For example, if f is a beam detection functionality, $\pi_{i,f,d}$ can represent the minimum detection accuracy of the model. Similarly, in the case of capacity forecasting, a request can specify a maximum tolerable Mean Square Error (MSE) level $\pi_{i,f,d}$. We do not make any assumptions on the physical meaning of $\pi_{i,f,d}$ as it reasonably differs from one functionality to the other. For example, if $\tau_{i,f,d} = 1$ and $\sigma_{m,f} = 1$ for a model m, such model can be instantiated on node d to deliver functionality f requested by i if and only if $\beta_{m,f,d} \geq \pi_{i,f,d}$.

• *Timing requirements.* In principle, all functionalities could be executed at the non-RT RIC by leveraging its centrality in the network and its ability to gather data over the O-RAN O1 interface. However, some functionalities might have strict latency requirements that make their execution at nodes far away from the location where the input is generated impractical or inefficient. For this reason, $\delta_{i,f,d} \geq 0$ represents the maximum latency request i can tolerate in executing f on d; For example, although instantiating a ML/AI model to execute beam prediction at the non-RT RIC might be possible, it does not guarantee real-time response times as IQ samples and other data necessary to control the transmission (or reception) beams must travel from the RUs throughout the entire network and reach the non-RT RIC, which inevitably incurs in high latency and violation of real-time requirements.

• *Data source.* For each request i, the NO also specifies the subset of nodes whose generated (or collected) data must be used to deliver functionality f on node d. This set is defined as $\mathcal{D}_i^{\text{IN}} = (\mathcal{D}_{i,f,d}^{\text{IN}})_{d\in\mathcal{D}_i, f\in\mathcal{F}_{i,d}}$, where $\mathcal{D}_{i,f,d}^{\text{IN}} \subseteq \mathcal{D}$. This information is paramount to ensure that each model is fed with the proper data generated by the intended sources only. For any tuple (i, f, d) we assume that $c_{d,d'} = 1$ for all $d' \in \mathcal{D}_{i,f,d'}^{\text{IN}}$, i.e., data collection from $\mathcal{D}_{i,f,d}^{\text{IN}}$ to node d is always feasible.

In the remaining of this paper, we use $\mathcal{I}$ to represent the set of outstanding requests with $I = |\mathcal{I}|$ being their number.

4.4 The Orchestration Engine

As depicted in Fig. 3, once requests are submitted to OrchestRAN, the next step consists in determining which requests can be accommodated and orchestrating the corresponding ML/AI models. This task is accomplished by the Orchestration Engine, which selects the most suitable models from the ML/AI Catalog and the location where they should execute (step I). Then, OrchestRAN embeds the models into containers (e.g., Docker containers of dApps, xApps, rApps) (step II) and dispatches them to the selected nodes (step III). Here, they are fed data from the RAN and execute their functionalities (step IV). The selection of the models and of their optimal execution location is performed by solving the *orchestration problem* discussed in detail in Sections 5 and 6. This results in an *orchestration policy*, which is converted into a set of O-RAN applications that are dispatched and executed at the designated network nodes, as discussed next.

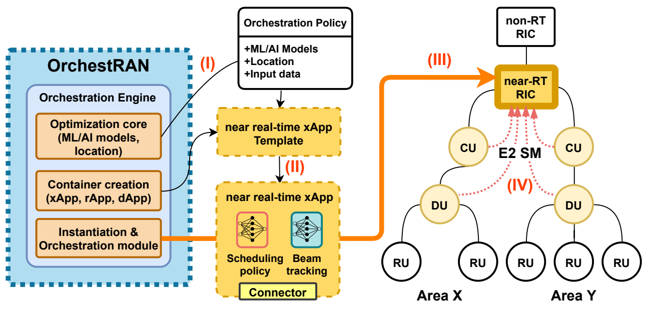

Figure 3. An example of creation and dispatchment of an xApp on the near-RT RIC via OrchestRAN.

4.4.1 Container creation

To embed models in different O-RAN applications, containers integrate two subsystems, which are automatically compiled from descriptive files upon instantiation. The first is the model itself, and the second is an application-specific *connector*. This is a library that interfaces with the node where the application is running (i.e., with the DU in the case of dApps, near-RT RIC for xApps, and non-RT RIC for rApps), collects data from $\mathcal{D}_i^{\text{IN}}$ and sends control commands to nodes in $\mathcal{D}_i$.

4.4.2 Dispatchment and Instantiation

Once the containers are generated, OrchestRAN dispatches them to the proper endpoints specified in the orchestration policy, where are instantiated and interfaced with the RAN to receive input data. For example, xApps automatically send an E2 subscription request to nodes in $\mathcal{D}_i^{\text{IN}}$, and use custom Service Models (SMs) to interact with them over the E2 interface [46] (see Fig. 3).

The implementation details of these aspects will be covered in detail in Section 8.

4.5 Additional OrchestRAN features

4.5.1 Functionality outsourcing

Any functionality that was originally intended to execute at node d' can be outsourced to any other node d'' as long as $c_{d',d''} = 1$. Accordingly, any node d'' that is connected to d', i.e., $c_{d',d''} = 1$, represents a possible candidate to outsource any ML/AI model that was intended to execute at d'. As we will discuss next, however, not all models can be outsourced to neighbor nodes. Indeed, the node hosting the outsourced model must have access to the required input data, have enough resources to instantiate and execute the outsourced model, and must satisfy performance and timing requirements of the original request.

Authorized licensed use limited to: Northeastern University. Downloaded on January 29,2024 at 14:47:29 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3342711

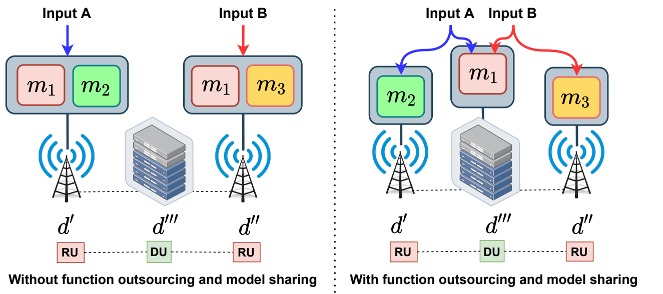

Figure 4. Example of function outsourcing and model sharing in Open RAN.

4.5.2 Model sharing

The limited amount of resources, especially at DUs and RUs, calls for efficient resource allocation strategies. If multiple requests involve the same functionalities on the same group of nodes, an efficient approach consists in deploying a single model that can be shared across all requests.

For the sake of clarity, in Fig. 4 (left) we show an example where a request can be satisfied by instantiating models m_1 and m_2 on d', and a second one that can be accommodated by instantiating models m_1 and m_3 on d''. Clearly, this deployment requires the instantiation of four models and, despite it satisfies the two requests, it is not the most efficient solution. Indeed, Fig. 4 (right) shows an alternative solution where m_1 (common to both requests) is *outsourced* to d''' and it is shared between the two requests, with a total of three deployed models, against the four required in Fig. 4 (left). Moreover, the energy and resource consumption of the two approaches is quite different: The approach on the right of Fig. 4 leverages a single instance of m_1, consuming $\rho^{\xi}_{m_1}$ for each resource of type ξ. The one on the left employs $2 \times \rho^{\xi}_{m_1}$ to instantiate and execute two instances of m_1, which results in a higher resource utilization footprint.

In Section 5.5, we also discuss those cases where model sharing and/or function outsourcing are nonviable.

5 The Orchestration Problem

Clearly, enabling network intelligence is not trivial and depends on many factors such as resources and input availability, latency requirements, but—most importantly—it requires orchestration capabilities to split data-driven functionalities across several points of the network. All of these aspects will be discussed in detail in the following.

Let $x^{i,f,d}_{m,k,d'} \in \{0,1\}$ be a binary variable such that $x^{i,f,d}_{m,k,d'} = 1$ if functionality f demanded by request i on node d is provided by instance k of model m instantiated on node d'. In the following, we refer to the variable $\mathbf{x} = (x^{i,f,d}_{m,k,d'})_{i,f,d,m,k,d'}$ as the *orchestration policy*, where $i \in \mathcal{I}, f \in \mathcal{F}, (d,d') \in \mathcal{D} \times \mathcal{D}, m \in \mathcal{M}, k = 1 \dots C_{m,d'}$.

5.1 Conflict avoidance

For any tuple (i,f,d) such that $\tau_{i,f,d} = 1$, we assume that OrchestRAN can instantiate at most one model to avoid multiple models controlling the same parameters and/or functionalities. As mentioned earlier, this can be achieved by either instantiating the model at d, or by outsourcing it to another node $d' \neq d$ such that $c_{d,d'} = 1$. The above requirement can be formalized as follows:

$$\sum_{m \in \mathcal{M}} \sigma_{m,f} \sum_{d' \in \mathcal{D}} \sum_{k=1}^{C_{m,d'}} c_{d,d'} x^{i,f,d}_{m,k,d'} = y_i \tau_{i,f,d} \qquad (1)$$

where $y_i \in \{0,1\}$ indicates whether or not i is satisfied. Specifically, (1) ensures that: (i) For any tuple (i,f,d) such that $\tau_{i,f,d} = 1$, function f is provided by one model only, and (ii) $y_i = 1$ (i.e., request i is satisfied) if and only if OrchestRAN deploys models providing all functionalities specified in $\mathcal{F}_i$.

5.2 Complying with the requirements

An important aspect of the orchestration problem is guaranteeing that the orchestration policy $\mathbf{x}$ satisfies the minimum performance requirements $\boldsymbol{\pi}_i$ of each request i, and that both data collection and execution procedures do not exceed the maximum latency constraint $\delta_{i,f,d}$. For example, one might specify that a specific functionality is delivered with a prediction accuracy that is above a desired threshold. Similarly, a request might also specify the time-scale at which a specific functionality should be executed (e.g., non-RT and in the order of a few seconds, or near-RT with a time-scale of a few milliseconds). These requirements are captured by the following constraints.

5.2.1 Quality of models

For each tuple (i,f,d) such that $\tau_{i,f,d} = 1$, NOs can specify a minimum performance level $\pi_{i,f,d}$. This can be enforced via the following constraint

$$\chi_{i,f,d} \sum_{m \in \mathcal{M}} \sum_{d' \in \mathcal{D}} \sum_{k=1}^{C_{m,d'}} c_{d,d'} x^{i,f,d}_{m,k,d'} A_{m,f,d} \geq \chi_{i,f,d} y_i \pi_{i,f,d} \qquad (2)$$

where $A_{m,f,d} = \beta_{m,f,d} \gamma_{m,f} \sigma_{m,f}$, and the performance score $\gamma_{m,f}$ is defined in Section 4.2. In (2), $\chi_{i,f,d} = 1$ if the goal is to guarantee a value of $\gamma_{m,f}$ higher than a minimum performance level $\pi_{i,f,d}$, and $\chi_{i,f,d} = -1$ if the goal is to keep $\gamma_{m,f}$ below a maximum value $\pi_{i,f,d}$.

5.2.2 Control-loop time-scales

Another important requirement of the orchestration problem is the time-scale at which each functionality must execute. Considering latency is essential as all functionalities and models must run within the required temporal window. To satisfy the second constraint, we must ensure that each O-RAN application can collect data to be fed to each model and compute an output within the required temporal window.

Each model m requires a specific type of input t^{IN}_m and, for each tuple (i,f,d), we must ensure that the time needed to collect such input from nodes in $\mathcal{D}^{\mathrm{IN}}_{i,f,d}$ does not exceed $\delta_{i,f,d}$. For each orchestration policy $\mathbf{x}$, the *data collection time* can be formalized as follows:

$$\Delta_{i,f,d}(\mathbf{x}) = \sum_{m \in \mathcal{M}} \sigma_{m,f} \sum_{d' \in \mathcal{D}} \sum_{k=1}^{C_{md'}} x^{i,f,d}_{m,k,d'} \sum_{d'' \in \mathcal{D}^{\mathrm{IN}}_{i,f,d}} c_{d',d''} \Theta^{i,f,d}_{m,d',d''} \qquad (3)$$

Authorized licensed use limited to: Northeastern University. Downloaded on January 29,2024 at 14:47:29 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3342711

where

$$\Theta^{i,f,d}_{m,d',d''} = \left(\frac{s_{t^{\text{IN}}_m}}{b_{d',d''} |\mathcal{D}^{\text{IN}}_{i,f,d}|} + T_{d',d''} \right), \tag{4}$$

$s_{t^{\text{IN}}_m}$ is the maximum input size of model m measured in bytes, $b_{d',d''}$ is the data rate of the link between nodes d'' and d', and $T_{d',d''}$ represents the propagation delay between nodes d' and d''. Note that since the link between nodes d' and d'' might involve several hops, $b_{d',d''}$ represents the bottleneck data rate over the entire routing path, which makes it possible to capture the actual data rate at which data will be sent from d'' to d' It is worth mentioning that the propagation delay and the data rate depend on network-wide routing policies, which might change over time as more nodes are deployed and routing policies are changed according to load balancing policies. These parameters, which might change over time, are updated periodically (e.g., via `qperf` and `iperf`). Moreover, although ML/AI models might ingest a variable input size, in this analysis we consider the worst case data collection time by considering the maximum size of the input which is a hard constraint especially to ensure that the RICs can perform inference within their maximum tolerable timescale.

Let T^{exec}_m be the time to execute model m on node d'. For any tuple (i, f, d), the *execution time* under orchestration policy $\mathbf{x}$ is

$$\Delta^{EXEC}_{i,f,d}(\mathbf{x}) = \sum_{m\in\mathcal{M}} \sigma_{m,f} \sum_{d'\in\mathcal{D}} T^{exec}_{m,d'} \sum_{k=1}^{C_{m,d'}} x^{i,f,d}_{m,k,d'} \tag{5}$$

By combining (3) and (5), any orchestration policy $\mathbf{x}$ must satisfy the following constraint for all (i, f, d) tuples:

$$\Delta_{i,f,d}(\mathbf{x}) + \Delta^{EXEC}_{i,f,d}(\mathbf{x}) \le \delta_{i,f,d}\ \tau_{i,f,d} \tag{6}$$

5.3 Avoiding resource over-provisioning

We must guarantee that the resources consumed by the O-RAN applications do not exceed the resources ρ^ξ_d of type ξ available at each node (i.e., ρ^ξ_d). For each node d and resource type ξ, we have

$$\sum_{m\in\mathcal{M}} \rho^\xi_m \sum_{k=1}^{C_{md}} z_{m,k,d} \le \rho^\xi_d \tag{7}$$

where $z_{m,k,d} \in \{0, 1\}$ indicates whether instance k of model m is associated to at least one model on node d. Specifically, let

$$n_{m,k,d} = \sum_{i\in\mathcal{I}} \sum_{f\in\mathcal{F}_i} \sum_{d'\in\mathcal{D}} x^{i,f,d'}_{m,k,d} \tag{8}$$

be the number of tuples (i, f, d') assigned to instance k of model m on node d ($n_{m,k,d} > 1$ implies that m is shared). Notice that (7) and (8) are coupled one to another as $z_{m,k,d} = 1$ if and only if $n_{m,k,d} > 0$. This conditional relationship can be formulated by using the following big-M formulation [51]

$$n_{m,k,d} \ge 1 - M(1 - z_{m,k,d}) \tag{9}$$
$$n_{m,k,d} \le M z_{m,k,d} \tag{10}$$

where M is a real-valued number whose value is larger than the maximum value of $n_{m,k,d}$, i.e., $M > IFD$ [51].

5.4 Problem formulation

For any request i, let $v_i \ge 0$ represent its value. The goal of OrchestRAN is to compute an orchestration policy $\mathbf{x}$ maximizing the total value of requests being accommodated by selecting (i) which requests can be accommodated; (ii) which models should be instantiated; and (iii) where they should be executed to satisfy request performance and timescale requirements. This can be formulated as

$$\max_{\mathbf{x},\mathbf{y},\mathbf{z}} \quad \sum_{i\in\mathcal{I}} y_i v_i \tag{11}$$
$$\text{subject to} \quad \text{Constraints (1), (2), (6), (7), (9), (10)}$$
$$x^{i,f,d}_{m,k,d'} \in \{0, 1\} \tag{12}$$
$$y_i \in \{0, 1\} \tag{13}$$
$$z_{m,k,d} \in \{0, 1\} \tag{14}$$

where $\mathbf{x}$ is the orchestration policy, $\mathbf{y} = (y_i)_{i\in\mathcal{I}}$ and $\mathbf{z} = (z_{m,k,d})_{m\in\mathcal{M},k=1,\dots,C_{m,d},d\in\mathcal{D}}$.

The objective of this problem is to satisfy as many requests as possible and maximize the associated value by enforcing operator intents while reducing overhead and resource utilization on the network via model sharing. A particularly relevant case is that where $v_i = 1$ for any request i, i.e., the goal of OrchestRAN is to maximize the number of satisfied requests.

5.5 Disabling model sharing

Model sharing allows a more efficient use of the available resources. However, it is also worth mentioning that, in some cases, model sharing and function outsourcing can be neither feasible nor possible. Indeed, although several requests from the same NO can share the same model, it is also true that different NOs might not be willing to share the same models out of business and privacy concerns. Similarly, although two nodes are connected one with another, exchanging information incur in additional latency depending on the available bandwidth on the link between the two nodes as well as the propagation delay. Such latency might exceed maximum latency requirements for real-time applications (in the order of a few milliseconds), thus making outsourcing inefficient and unpractical.

In this case, model sharing can be disabled in OrchestRAN by guaranteeing that a model is assigned to one request only. This is achieved by adding the following constraint for any model m, node d' and $k = 1, .., C_{m,d'}$

$$\sum_{i\in\mathcal{I}} \sum_{d\in\mathcal{D}} \sum_{f\in\mathcal{F}_{i,d}} x^{i,f,d}_{m,k,d'} \le 1 \tag{15}$$

5.6 NP-hardness of the Orchestration Problem

Although the utility function and constraints are linear, solving the orchestration problem is not trivial as the optimization variables are binary. Indeed, Problem (11) is a Binary Integer Linear Programming (BILP) problem which can be shown to be NP-hard. The proof consists in building a polynomial-time reduction of the 3-SAT problem (which is NP-complete) to an instance of Problem (11) [52].

Authorized licensed use limited to: Northeastern University. Downloaded on January 29,2024 at 14:47:29 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3342711

6 Solving the Orchestration Problem

BILP problems such as Problem (11) can be optimally solved via Branch-and-Bound (B&B) techniques [53], readily available within well-established numerical solvers, e.g., CPLEX, MATLAB, Gurobi. However, due to the extremely large number N_{OPT} of optimization variables, these solvers might still fail to compute an optimal solution in a reasonable amount of time, especially in large-scale deployments. Indeed, $N_{\mathrm{OPT}} = |\mathbf{x}| + |\mathbf{y}| + |\mathbf{z}| \approx |\mathbf{x}|$, where $|\mathbf{x}| = \mathcal{O}(IFD^2MC_{\max})$, $|\mathbf{y}| = \mathcal{O}(I)$, $|\mathbf{z}| = \mathcal{O}(MDC_{\max})$, and $C_{\max} = \max_{m\in\mathcal{M},d\in\mathcal{D}}\{C_{m,d}\}$. For example, a deployment with $D = 20$, $M = 13$, $I = 10$, $F = 7$ and $C_{\max} = 3$ involves $\approx 10^6$ optimization variables.

6.1 Combating Dimensionality via Variable Reduction

To mitigate the "curse of dimensionality" of the orchestration problem, we have developed two pre-processing algorithms to reduce the complexity of Problem (11) while guaranteeing the optimality of the computed solutions. We leverage a technique called *variable reduction* [54]. This exploits the fact that, due to constraints and structural properties of the problem, there might exist a subset of *inactive* variables whose value is always zero. For example, if a request i requires function f at node d, we cannot satisfy this request by selecting a model m which does not offer f, i.e., $\sigma_{m,f} = 0$. Indeed, selecting a model that is not well-suited to satisfy a function requirement for a given request automatically results in an unfeasible solution, which should be avoided and removed.

These variables do not participate in the optimization process, yet they increase its complexity. To identify those variables, we have designed the following two techniques.

- *Function-aware Pruning (FP).* It identifies the set of inactive variables
$$\mathbf{x}_{-}^{\mathrm{FP}} = \{x_{m,k,d'}^{i,f,d} : \tau_{i,f,d} = 0 \vee \sigma_{m,f} = 0, \forall i \in \mathcal{I}, f \in \mathcal{F}, (d,d') \in \mathcal{D}\times\mathcal{D}, m \in \mathcal{M}, k = 1,\dots,C_{,m,d}\}, \quad (16)$$
which contains all the $x_{m,k,d}^{i,f,d}$ variables such that either (i) $\tau_{i,f,d} = 0$, i.e., request i does not require function f at node d, or (ii) $\sigma_{m,f} = 0$, i.e., model m does not offer function f;
- *Architecture-aware Pruning (AP).* This procedure identifies all of the variables whose activation results in instantiating a model on a node that cannot receive input data from nodes in $\mathcal{D}_{i,f,d}^{\mathrm{IN}}$. Indeed, for a given tuple (i,f,d) such that $\tau_{i,f,d} = 1$, we cannot instantiate any model on a node d' such that $c_{d,d'} = 0$, i.e., the two nodes are not connected. The set of these inactive variables is defined as
$$\mathbf{x}_{-}^{\mathrm{AP}} = \{x_{m,k,d'}^{i,f,d} : c_{d,d'} = 0, \forall i \in \mathcal{I}, f \in \mathcal{F}, (d,d') \in \mathcal{D}\times\mathcal{D}, m \in \mathcal{M}, k = 1,\dots,C_{,m,d}\}. \quad (17)$$

Once we have identified all inactive variables, Problem (11) is cast into a lower-dimensional space where the new set of optimization variables is equal to $\tilde{\mathbf{x}} = \mathbf{x} \setminus \{\mathbf{x}_{-}^{\mathrm{FP}} \cup \mathbf{x}_{-}^{\mathrm{AP}}\}$.

Remark. It is important to note that inactive variables either lead to unfeasible orchestration strategies or are insufficient to fulfill any request. As a consequence, those variables will always be set to zero. Therefore, both the AP and FP preprocessing techniques described above reduce the number of problem variables while still ensuring the optimality of the computed solution [54]. Note that the activation of any previously inactive variable, cannot in any way contribute to the utility function $\sum_{i\in\mathcal{I}} y_i v_i$, but only results in constraint violations. This is due to the fact that $y_i = 1$ if and only if request i is satisfied and the solution is feasible. Both of these conditions are unattainable by inactive variables which, as described above, either lead to unfeasible solutions or are insufficient to satisfy any request. The impact of these procedures on the complexity of the orchestration problem will be investigated in Section 7.

6.2 Graph Tree Branching

To further reduce the complexity of the problem, in this section we present a solution to Problem (11) that leverages the tree structure of the RAN infrastructure to split the optimization problem into smaller instances that are solved independently.

Notice that $|\mathbf{x}| = \mathcal{O}(IFD^2MC_{\max})$, i.e., the number of variables of the orchestration problem grows quadratically in the number D of nodes. This becomes critical when a few tens of RICs are expected to control and interact with tens of thousands of RUs, DUs and CUs deployed across multiple regions. Since the majority of nodes of the infrastructure are RUs, DUs and CUs, it is reasonable to conclude that these nodes are the major source of complexity. Moreover, O-RAN systems operate following a cluster-based approach where each near-RT RIC controls a subset of CUs, DUs and RUs of the network only, i.e., a *cluster*, which have no (or limited) interactions with nodes from other clusters.

These two intuitions are the rationale behind the low-complexity and scalable solution proposed in this section, which consists in splitting the infrastructure tree into smaller subtrees—each operating as an individual cluster—and creating sub-instances of the orchestration problem that only accounts for requests and nodes regarding the considered subtree. The main steps of this algorithm are:

- *Step I*: Let C be the number of near-RT RICs in the non-RT RIC domain. For each cluster c, the c-th subtree $\mathcal{D}_c \subseteq \mathcal{D}$ is defined such that $\mathcal{D} = \bigcup_{c=1}^{C}\mathcal{D}_c$ and $\bigcap_{c=1}^{C}\mathcal{D}_c = d^{\mathrm{root}}$, with d^{root} being the non-RT RIC. A variable $\alpha_{d,c} \in \{0,1\}$ is used to determine whether a node d belongs to cluster c or not. Specifically, $\alpha_{d,c} = 1$ if the node belongs to cluster c, $\alpha_{d,c} = 0$ otherwise. Since, $\bigcap_{c=1}^{C}\mathcal{D}_c = d^{\mathrm{root}}$, we have that $\sum_{c=1}^{C}\alpha_{d,c} = 1$ for any $d \in \mathcal{D}\setminus\{d^{\mathrm{root}}\}$;
- *Step II*: For each subtree $\mathcal{D}_c$ we identify the subset $\mathcal{I}_c \subseteq \mathcal{I}$ such that $\mathcal{I}_c = \{i \in \mathcal{I} : \sum_{f\in\mathcal{F}}\sum_{d\in\mathcal{D}_c}\tau_{i,f,d} \geq 1\}$ contains all the requests that involve nodes belonging to cluster c only;
- *Step III*: We solve Problem (11) via B&B considering only requests in $\mathcal{I}_c$ and nodes in $\mathcal{D}_c$. The solution is a tuple $(\mathbf{x}_c, \mathbf{y}_c, \mathbf{z}_c)$ specifying which models are instantiated and where $(\mathbf{x}_c)$, which requests are satisfied in cluster c $(\mathbf{y}_c)$ and what instances of the models are instantiated on each node of $\mathcal{D}_c$ $(\mathbf{z}_c)$.

Remark. This branching procedure might compute solutions with *partially satisfied requests*. These are requests that

Authorized licensed use limited to: Northeastern University. Downloaded on January 29,2024 at 14:47:29 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3342711

are accommodated on a subset of clusters only, which violates Constraint 1. However, as we will show in Section 7, this procedure is scalable as each subtree $\mathcal{D}_c$ involves a limited number of nodes only, and we can solve each lower-dimensional instance of Problem (11) in parallel and in less than 0.1 s.

The procedures involved in the computation of the solution to the orchestration problem are summarized as pseudo-code in Algorithm 1.

Algorithm 1: Orchestration Engine pseudo-code

```
Result: A solution tuple $(\mathbf{x}, \mathbf{y}, \mathbf{z})$
while True do
    Wait to collect requests from NOs;
    if Complexity reduction needed then
        Apply FP, AP and/or Graph Tree Branching;
    end
    Generate problem instance;
    Solve problem and compute $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ via
      Branch-and-Bound;
    Generate rApp, xApp, dApp containers;
    Dispatch and instantiate containers;
end
```

7 Numerical Evaluation

To evaluate the performance of OrchestRAN in large-scale scenarios, we have developed a simulation tool in MATLAB that uses CPLEX to execute optimization routines. For each simulation, NOs submit $R = 20$ randomly generated requests, each specifying multiple sets of functionalities and nodes, as well as the desired timescale. Unless otherwise stated, we consider a single-domain deployment with 1 non-RT RIC, 4 near-RT RICs, 10 CUs, 30 DUs and 90 RUs. For each simulation, the number of network nodes is fixed, but the tree structure of the infrastructure is randomly generated. We consider the three cases shown in Table 2, where we limit the type of nodes that can be included in each request. Similarly, we also consider the three cases in Table 3. For each case, we specify the probability that the latency requirement $\delta_{i,f,d}$ for each tuple (i, f, d) is associated to a specific timescale. The combination of these 6 cases covers relevant Open RAN applications.

The ML/AI Catalog consists of $M = 13$ models that provide $F = 7$ different functionalities. Ten models use metrics from the RAN (e.g., throughput and buffer measurements) as input, while the remaining three models are fed with IQ samples from RUs. The input size $s_{t_m^{\text{IN}}}$ is set to 100 and 1000 bytes for the metrics and IQ samples, respectively. For the sake of illustration, we assume that $\beta_{m,f,d} = \sigma_{m,f}$, $\pi_{i,f,d} = \tau_{i,f,d}$ and $C_{m,d} = 3$ for all $m \in \mathcal{M}$, $i \in \mathcal{F}$, $f \in \mathcal{F}$ and $d \in \mathcal{D}$. The execution time of each model is equal across all models and nodes and set to $T^{exec}_{m,d} = 1$ ms. The available bandwidth $b_{d,d'}$ is 100 Gbps between non-RT RIC and near-RT RIC, 50 Gbps between near-RT RICs and CUs, 25 Gbps between CUs and DUs, and 20 Gbps between DUs and RUs, while the propagation delay $T_{d,d'}$ is set to $[10, 10, 5, 1]$ ms, respectively. The resources ρ_d available at each node are represented by the number of available CPU

Table 2
Controllable nodes.

Case \ Requested Nodes	non-RT RIC	near-RT RIC	CU	DU	RU
All nodes (ALL)	✓	✓	✓	✓	✓
Edge and RAN (ER)	✗	✓	✓	✓	✓
RAN only (RO)	✗	✗	✓	✓	✓

Table 3
Request timescale cases and probabilities.

Case \ Time scale	TTI-level - $\leq$ 0.01s	Sub-second - $\leq$ 1s	Long - $>$ 1s
Delay-Tolerant (DT)	0.2	0.2	0.6
Low Latency (LL)	0.2	0.6	0.2
Ultra-Low Latency (ULL)	0.6	0.4	0

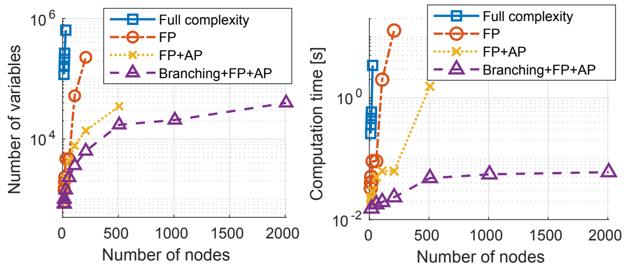

Figure 5. Number of variables and computation time for different network size.

cores, and we assume that each model requests one core only, i.e., $\rho_m = 1$. The number of cores available at non-RT RICs, near-RT RICs, CUs, DUs and RUs are 128, 8, 4, 2, and 1, respectively. Results presented in this section are averaged over 100 independent simulation runs.

• *Computational complexity.* We start our analysis by presenting results on the computational complexity of OrchestRAN and of the optimization algorithms of Section 6. Fig. 5 shows the number of optimization variables and computation time of our algorithms with varying network size. At each simulation run, we consider a single non-RT RIC and a randomly generated tree graph that matches the considered size. As expected, the number of variables and the complexity increase with larger networks. This can be mitigated by using our FP and AP pre-processing algorithms, which reduce the number of optimization variables while ensuring the optimality of the computed solution. Their combination allows computation of optimal solutions in 0.1 s and 2 s for networks with 200 and 500 nodes, respectively. Fig. 5 also shows the benefits of branching the optimization problem into sub-problems of smaller size (Section 6.2). Although the branching procedure might produce partially satisfied requests, it results in a computation time lower than 0.1 s even for instances with 2000 nodes, providing a fast and scalable solution for large-scale applications.

• *Acceptance ratio.* Fig. 6 (left) shows the acceptance ratio for different cases and algorithms. The number of accepted requests decreases when moving from loose timing requirements (i.e., Delay-tolerant (DT)), to tighter ones (i.e., Low Latency (LL) and Ultra-Low Latency (ULL)). For example, while 95% of requests are satisfied on average for the DT configuration, we observe ULL instances in which only 70%

Authorized licensed use limited to: Northeastern University. Downloaded on January 29,2024 at 14:47:29 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3342711

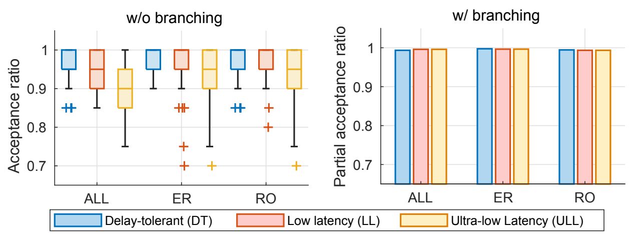

Figure 6. (Left) Ratio of accepted requests w/ model sharing but w/o branching; (Right) Ratio of partially accepted requests w/ model sharing and branching.

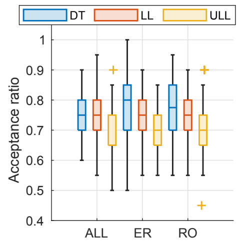

Figure 8. Percentage of accepted requests w/o model sharing for different cases.

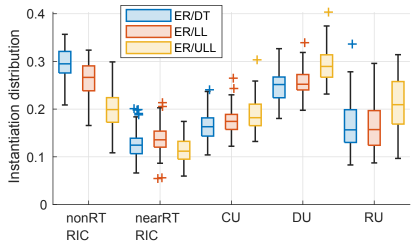

Figure 9. Distribution of model instantiation for different cases.

of requests are accepted. Indeed, TTI-level services may only be possible at the DUs/RUs which, however, have limited resources and cannot support the execution of many concurrent O-RAN applications. In Fig. 6 (right), we show the probability that a request is partially accepted when considering the branching algorithm. Specifically, it shows that branching results in ≈99% of requests being partially satisfied on one subtree or more. This means that in the case where not enough resources are available to accept the entirety of the request, OrchestRAN can satisfy portions of it. Thus, requests that would be otherwise rejected can be at least partially accommodated.

• *Advantages of model sharing.* Fig. 7 shows the resource utilization with and without model sharing (left) and the corresponding resource utilization saving (right). As expected, model sharing always results in lower resource utilization and uses 2× less resources than the case without model sharing. Fig. 8 shows the acceptance ratio when model sharing is disabled, and by comparing it with Fig. 6 (left)—where model sharing is enabled—we notice that model sharing also increases the acceptance ratio. Specifically, model sharing accommodates at least 90% of requests in all cases, while this number drops to ≈70% when model sharing is disabled.

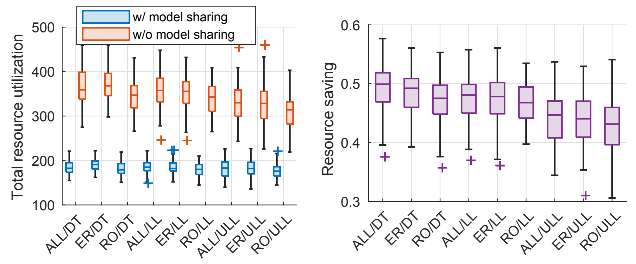

Figure 7. Resource utilization and saving with and without model sharing.

To better understand how OrchestRAN orchestrates intelligence, Fig. 9 shows the distribution of models across the different network nodes for the ER case (see Table 2) with different timing constraints. Requests with loose timing requirements (DT) result in ≈45% of models being allocated in the RICs. Instead, stringent timing constraints (LL and ULL) result in ≈70% of models being instantiated at CUs, DUs, and RUs.

In Fig. 10, we show the percentage of shared models under the same configurations considered in Fig. 7. We show that in most cases, more than 30% of the ML/AI models can be shared (assuming that all requests support sharing), with peaks of 55% sharing ratio in the DT case, which is tolerant to latency introduced by models being shared and deployed at higher levels of the O-RAN architecture (e.g., non-RT RIC).

• *Model outsourcing.* Finally, we investigate how OrchestRAN perfirms model outsourcing (e.g., instantiating a ML/AI model on a node different from the one requested by the NO while maintaining compliance with timing and controllability constraints). In Fig. 11, we report the ratio of ML/AI models being instantiated at levels of the network that are different from the ones originally requested by the NOs in the ULL case (i.e., NOs request ML/AI models that perform inference and control with time scales lower than 10 ms). Our results show that the majority of the ML/AI models is instantiated at levels that are lower than the requested one, with only a minimal portion being instantiated at levels that are hierarchically higher. Intuitively, since most requests demand for real-time ML/AI, OrchestRAN decides to instantiate the intelligence closer to the edge, so as to meet latency requirements and satisfy NOs requests.

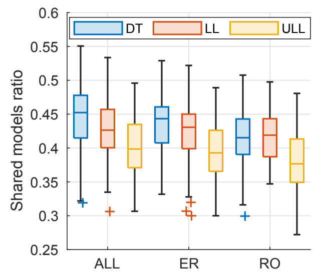

Figure 10. Percentage of shared ML/AI models for different cases.

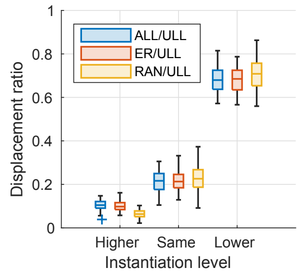

Figure 11. Displacement ratio for different cases in the ULL scenario.

In Fig. 12, we instead show the probability that a model requested at a certain node is outsourced to another node at a higher or lower hierarchy level.

We notice that in the ULL case (bottom part of the figure), the probability of instantiating a model at RAN nodes is high with CUs and DUs being the preferred nodes to instantiate ML/AI models to meet the stringent timing requirements. On the contrary, in the DT case (top part of the figure), we see that a higher percentage of models are instantiated on the RICs even if they are being requested to execute at the RAN (e.g., ALL/DT and ER/DT). These results show that, in general, when the timing constraints are stringent, model outsourcing is unlikely to happen. On the contrary, when the timing constraints are leaning more

Authorized licensed use limited to: Northeastern University. Downloaded on January 29,2024 at 14:47:29 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3342711

toward non-real-time execution (e.g., DT case) the RICs are preferred locations ot accommodate models thanks to their higher resource availability.

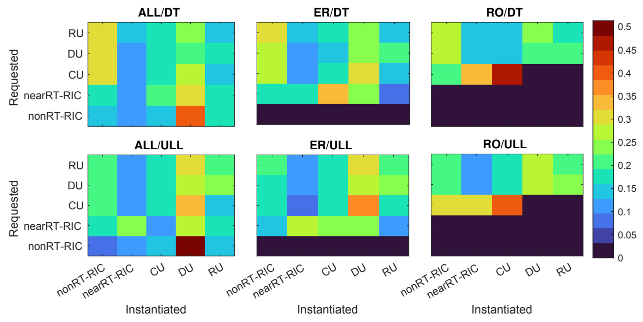

Figure 12. Model outsourcing analysis for different cases.

8 Prototype implementation

To demonstrate the effectiveness of OrchestRAN, we leveraged OpenRAN Gym [55], an O-RAN-compliant and open source large-scale experimental platform developed on top of the Colosseum wireless network emulator [29]. Colosseum includes 128 computing servers (i.e., Standard Radio Nodes (SRNs)), each controlling a USRP X310 Software-defined Radio (SDR), and a Massive Channel Emulator (MCHEM) emulating wireless channels between the SRNs to reproduce realistic and time-varying wireless characteristics (e.g., path-loss, multi-path) under different deployments (e.g., urban, rural, etc.) [29].

We leverage the publicly available tool SCOPE [34] to instantiate a softwarized cellular network with 7 base stations and 42 User Equipments (UEs) (6 UEs per base station) on the Colosseum city-scale downtown Rome scenario, and to interface the base stations with the O-RAN near-RT RIC through the E2 interface. SCOPE, which is based on srsRAN [56], implements open Application Programming Interfaces (APIs) to reconfigure the base station parameters (e.g., slicing resources, scheduling policies, etc.) from O-RAN applications through closed-control loops, and to automatically generate datasets from RAN statistics (e.g., throughput, buffer size, etc.). Users are deployed randomly and generate traffic belonging to 3 different network slices configured as follows: (i) slice 0 is allocated an Enhanced Mobile Broadband (eMBB) service, in which each UE requests 4 Mbps constant bitrate traffic; (ii) slice 1 a Machine-type Communications (MTC) service, in which each UE requests Poisson-distributed traffic with an average rate of 45 kbps, and (iii) slice 2 to a Ultra Reliable and Low Latency Communication (URLLC) service, in which each UE requests Poisson-distributed traffic with an average rate of 90 kbps. We assume 2 UEs per slice, whose traffic is handled by the base stations, which use a 10 MHz channel bandwidth with 50 Physical Resource Block (PRB).

The high-level architecture of the OrchestRAN prototype on Colosseum, based on the OpenRAN Gym framework [55], is shown in Fig. 13. OrchestRAN runs in an Linux Container (LXC) embedding the components of Fig 2. For each experiment, we randomly generate a new set of requests every 4 minutes. The Orchestration Engine computes the optimal orchestration policy and embeds the models within O-RAN applications, e.g., xApp and dApps, that are dispatched to the nodes where they are executed. Specifically, xApps are instantiated on the ColO-RAN near-RT RIC [28] that runs in the form of Docker containers inside one of Colosseum SRNs. By leveraging the O-RAN E2 termination, these xApps interface with the RAN base stations to receive periodic KPIs and send the computed control actions. Similarly, dApps are instantiated on the cellular base station, implemented on Colosseum through the SCOPE framework [34]. After their instantiation, these applications interface with the underlying cellular protocol stack to take some control actions based on the run-time KPIs of the softwarized base station, which provides service to cellular UEs—implemented as additional SRNs on Colosseum—through the wireless scenarios emulated by MCHEM (see Fig. 13).

8.1 ML/AI Catalog creation

Since the focus of this paper is on orchestrating network intelligence rather than generating the actual ML/AI models that will be used for inference and control of the network, in this paper we use as a reference the set of ML/AI models developed and described in great details in our previous work [28]. These models have been trained via datasets collected via SCOPE [34] on Colosseum [29]. The datasets used in this paper are described in detail in [28], which also include a link to the publicly available repository that can be downloaded from GitHub.

In this work, we consider four ML/AI models that let us cover both prediction and control tasks. Models $M1$ and $M2$ have been trained to forecast the evolution of throughput and transmission buffer size KPIs. Both models use the same Downlink (DL) architecture that consists of a neural network with three fully connected layers with ReLu activation function. The first layer has a tunable number of neurons depending on the length of the input that will be used to predict the next sample. In this case, both $M1$ and $M2$ forecast the next measurement by processing an input represented by a one-dimensional array of size 10. Therefore, the first layer has 10 neurons, the last layer has 1 neuron and the intermediate layer has 64 neurons. $M1$ and $M2$ are both trained using the Adam optimizer, a Mean Squared Loss function and a learning rate of 0.001 with batch size of 32.

Table 4
DRL agents in the ML/AI Catalog.

	Reward			
	Slice 0	Slice 1	Slice 2	Actions
M3	max(Throughput)	max(TX pkts)	max(PRB ratio)	Scheduling
M4	max(Throughput)	max(TX pkts)	min(Buffer size)	Scheduling, RAN slicing

Models $M3$ and $M4$ control the parameters of the network to maximize different rewards through Proximal Policy Optimization (PPO)-based DRL agents (see Table 4). Specifically, $M3$ consists of three DRL agents, each making decisions on the scheduling policies of one slice only. The three agents aim at maximizing the throughput of slice 0, the number of transmitted packets of slice 1, and the ratio

Authorized licensed use limited to: Northeastern University. Downloaded on January 29,2024 at 14:47:29 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3342711

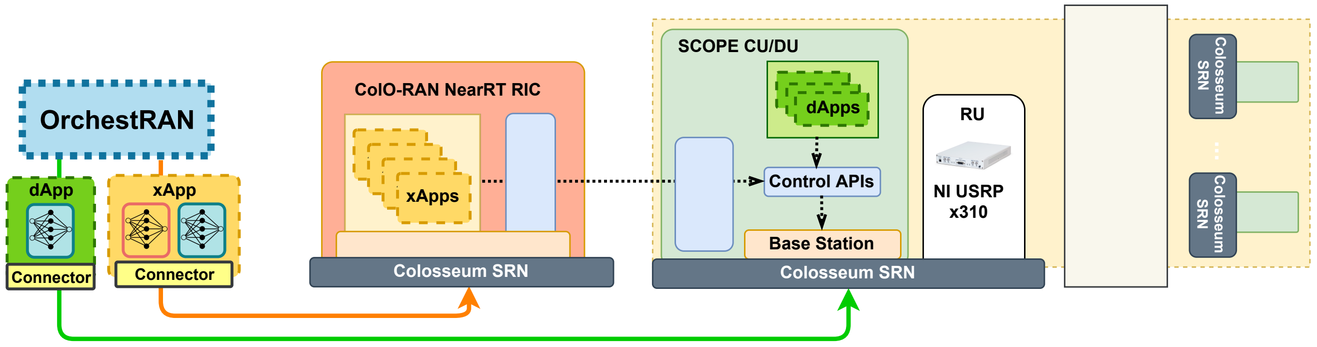

Figure 13. OrchestRAN prototype architecture as implemented on Colosseum.

between the allocated and requested PRBs (i.e., the *PRB ratio* which takes values in $[0, 1]$) of slice 2, respectively. Model $M4$, instead, consists of a single DRL agent controlling the scheduling and RAN slicing policies (i.e., how many PRBs are assigned to each slice) to *jointly* maximize the throughput of slice 0 and the number of transmitted packets of slice 1, and to minimize the buffer size of slice 2. Both models $M3$ and $M4$ follow an actor-critic architecture where the actor takes decisions based on the current state of the network while, during training, the critic is used to score the actions taken by the actor and orient the latter toward actions that yield higher rewards. Both actor and critic are implemented via fully connected neural networks with 5 layers with 30 neurons each. We refer the reader to [28] for more details on the training and design of the DRL agents used in $M3$ and $M4$.

In all cases, the ML/AI models have been trained to account for missing and/or incomplete data, for example, due to low traffic load or varying number of UEs. In our case, this is prevented by using a zero padding strategy which is also used during the training process to emulate the case of input data with a size that is smaller then what is required by the ML/AI model.

We consider the case where each model requires one CPU core, as well as three different configurations: (i) "RIC only", in which models can be executed via xApps at the near-RT RIC only; (ii) "RIC + lightweight DU", in which DUs have 2 cores each to execute up to two dApps concurrently; and (iii) "RIC + powerful DU", in which DUs are equipped with 8 cores. In all cases, the near-RT RIC has access to 50 cores. Overall, we ran more than 95 hours of experiments on Colosseum.

8.2 Orchestration policy definition

Once the orchestration policy $\mathbf{x}$ has been computed by OrchestRAN, it gets converted automatically into a JSON formatted file that specifies the ML/AI models to be deployed (through the `model_name` keyword), where they should execute (`node_name`), and which nodes will provide input data to the different models (`input_device_id`). Listing 1 shows an example of an orchestration policy that includes one xApp embedding model $M4$ (see Table 4) to be executed on the near-RT RIC to control DU 1, and a dApp embedding model $M3$ to be executed on DU 7 and locally control its scheduling decisions. The JSON file is parsed by OrchestRAN that leverages the above keywords to perform

```json
{
  "policy": [
    {
      "model_name": "M4",
      "node_name": "near_rt_ric",
      "input_device_id": "DU 1"
    },
    {
      "model_name": "M3",
      "node_name": "DU 7",
      "input_device_id": "DU 7"
    }
  ]
}
```

Listing 1. An example of an orchestration policy in JSON format including one xApp and one dApp.

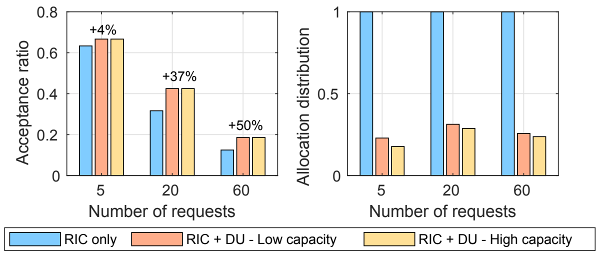

Figure 14. (Left) Ratio of accepted requests for different configurations; (right) probability of instantiating O-RAN applications at the near-RT RIC

the actual deployment on the requested nodes. As an example, the values associated to the `model_name` keyword (e.g., M3 and M4) are replaced with the Docker image of the specific models, and the values of the `input_device_id` keyword are passed to the xApp/dApp Docker containers that subscribe to the appropriate nodes. Finally, the commands to instantiate the requested xApps and dApps are issued by OrchestRAN on the nodes specified through the `node_name` keyword.

9 Experimental Results

In Fig. 14 (left), we show the ratio of requests accepted by OrchestRAN in our prototype. When all requests can be satisfied via xApps and rApps only (i.e., "RIC only" case), the acceptance ratio is lower than the case where dApps can also be used to execute such intelligence. Specifically, introducing dApps via DUs capable of executing ML/AI models to reconfigure their protocol stack results in an increase in acceptance ratio ranging from 4% in the "RIC

Authorized licensed use limited to: Northeastern University. Downloaded on January 29,2024 at 14:47:29 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3342711

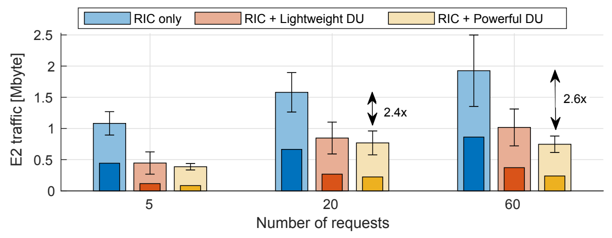

Figure 15. Traffic over O-RAN E2 interface for different configurations. Dark bars represent traffic related to payload only.

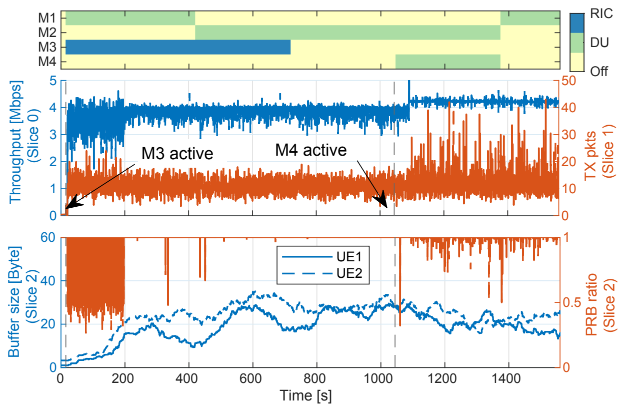

Figure 16. (Top) Dynamic activation of O-RAN applications at near-RT RIC and DU 7; (Center and bottom) Performance comparison for different deployments of O-RAN applications and network slices. Solid lines and dashed lines refer to traffic for $\mathrm{UE}_{1,i}$ and $\mathrm{UE}_{2,i}$ of Slice i.

only" case to 50% in the "RIC + powerful DU" case. These results show that extending intelligence to the very edge of the network makes it possible to offload control and inference tasks to the RAN and accommodate more requests from the NOs.

Fig. 14 (right) shows the probability that models are executed at the near-RT RIC for different configurations and number of requests. As expected, in the "RIC only" case, all models execute as xApps at the near-RT RIC, while both "RIC + lightweight DU" and "RIC + powerful DU" cases result in ≈25% of models executing at the RIC. The remaining 75% of the models are executed as dApps at the DUs. Fig. 15 shows the traffic in Mbyte over the E2 interface between the near-RT RIC and the DUs for the different configurations. This includes messages to set up the initial subscription between the near-RT RIC and the DUs, messages to report metrics from the DUs to the RIC (e.g., throughput, buffer size), and control messages from the RIC to the DUs (e.g., to update scheduling and RAN slicing policies). Results clearly show that ≈40% of the E2 traffic transports payload information (dark bars), while the remaining 60% consists of overhead data. Although the initial subscription messages exchanged between the near-RT RIC and the DUs are sent in all considered cases, running models as dApps at the DUs still results in up to 2.6× less E2 traffic if compared to the "RIC only" case.

Finally, we showcase the impact of the real-time execution of OrchestRAN on the network performance. We focus on DU 7, and in Fig. 16 (top) we show the location and time instant at which OrchestRAN instantiates the four models on the near-RT RIC and on base station 7 (i.e., referred to DU 7 for simplicity) for a single experiment. The impact on the network performance of the different orchestration policies is shown in Fig. 16 (center and bottom). Since $M1$ and $M2$ perform forecasting tasks only, the figure only reports the evolution of the metrics used to reward the DRL agents $M3$ and $M4$ (see Table 4) for different slices. We notice that OrchestRAN allows the seamless instantiation of dApps and xApps, controlling the same DU without causing any service interruptions. Moreover, although $M3$ and $M4$ share the same reward for slices 0 and 1, $M4$ can also make decisions on the network slicing policies. Thus, it provides a higher throughput for slice 0 (≈10% higher than $M3$), and a higher number of transmitted packets for slice 1 (≈2× higher than $M3$) (Fig. 16 (center)). Similarly, in the case of slice 2, $M3$ aims at maximizing the PRB ratio, while $M4$ at minimizing the size of the transmission buffer, which results in $M3$ and $M4$ computing different control policies for slice 2. As shown Fig. 16 (bottom), although $M3$ converges to a stable control policy that results in a PRB ratio ≈1, its buffer size is higher than that of $M4$. Conversely, the buffer size of slice 2 decreases once $M4$ is instantiated with a decrease in the PRB ratio.

10 Conclusions

In this paper, we presented OrchestRAN, a novel network intelligence orchestration framework for Open RAN systems. OrchestRAN is based upon O-RAN specifications and leverages the RIC xApps and rApps and O-RAN open interfaces to provide NOs with an automated orchestration tool for deploying data-driven inference and control solutions with diverse timing requirements. OrchestRAN has been equipped with orchestration algorithms with different optimality/complexity trade-offs to support non-RT, near-RT and RT applications. We assessed OrchestRAN performance and presented an O-RAN-compliant prototype by instantiating a cellular network with 7 base stations and 42 UEs on the Colosseum network emulator. Our experimental results demonstrate that OrchestRAN achieves seamless instantiation of O-RAN applications at different network nodes and time scales, and reduces the message overhead over the O-RAN E2 interface by up to 2.6× when instantiating intelligence at the edge of the network.

References

[1] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "Open, Programmable, and Virtualized 5G Networks: State-of-the-Art and the Road Ahead," *Computer Networks*, vol. 182, pp. 1–28, Dec. 2020.

[2] O-RAN WG1, "O-RAN Architecture Description - v2.00," Technical Specification, 2020.

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

[4] J. Du, C. Jiang, J. Wang, Y. Ren, and M. Debbah, "Machine Learning for 6G Wireless Networks: Carrying Forward Enhanced Bandwidth, Massive Access, and Ultrareliable/Low-Latency Service," *IEEE Vehicular Technology Magazine*, vol. 15, no. 4, pp. 122–134, 2020.

[5] S. Shen, T. Zhang, S. Mao, and G.-K. Chang, "DRL-Based Channel and Latency Aware Radio Resource Allocation for 5G Service-Oriented RoF-mmWave RAN," *Journal of Lightwave Technology*, 2021.

Authorized licensed use limited to: Northeastern University. Downloaded on January 29,2024 at 14:47:29 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and
content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3342711

[6] A. Okic, L. Zanzi, V. Sciancalepore, A. Redondi, and X. Costa-Pérez, "π-ROAD: a Learn-as-You-Go Framework for On-Demand Emergency Slices in V2X Scenarios," in *Proc. of IEEE INFOCOM*, 2021.

[7] D. Bega, M. Gramaglia, A. Garcia-Saavedra, M. Fiore, A. Banchs, and X. Costa-Perez, "Network Slicing Meets Artificial Intelligence: An AI-based Framework for Slice Management," *IEEE Communications Magazine*, vol. 58, no. 6, pp. 32–38, 2020.

[8] L. Bonati, S. D'Oro, M. Polese, S. Basagni, and T. Melodia, "Intelligence and Learning in O-RAN for Data-driven NextG Cellular Networks," *IEEE Communications Magazine*, vol. 59, no. 10, pp. 21–27, October 2021.

[9] S. Bakri, P. A. Frangoudis, A. Ksentini, and M. Bouaziz, "Data-Driven RAN Slicing Mechanisms for 5G and Beyond," *IEEE Trans. on Network and Service Management*, 2021.

[10] N. Salhab, R. Langar, R. Rahim, S. Cherrier, and A. Outtagarts, "Autonomous Network Slicing Prototype Using Machine-Learning-Based Forecasting for Radio Resources," *IEEE Communications Magazine*, vol. 59, no. 6, pp. 73–79, 2021.

[11] J. Mei, X. Wang, K. Zheng, G. Boudreau, A. B. Sediq, and H. Abouzeid, "Intelligent Radio Access Network Slicing for Service Provisioning in 6G: A Hierarchical Deep Reinforcement Learning Approach," *IEEE Trans. on Communications*, 2021.

[12] J. A. Ayala-Romero, A. Garcia-Saavedra, M. Gramaglia, X. Costa-Perez, A. Banchs, and J. J. Alcaraz, "vrAIn: Deep Learning based Orchestration for Computing and Radio Resources in vRANs," *IEEE Trans. on Mobile Computing*, 2020.

[13] L. Bonati, S. D'Oro, L. Bertizzolo, E. Demirors, Z. Guan, S. Basagni, and T. Melodia, "CellOS: Zero-touch Softwarized Open Cellular Networks," *Computer Networks*, vol. 180, pp. 1–13, Oct. 2020.

[14] B. Casasole, L. Bonati, S. D'Oro, S. Basagni, A. Capone, and T. Melodia, "QCell: Self-optimization of Softwarized 5G Networks through Deep Q-learning," in *Proc. of IEEE GLOBECOM*, 2021.

[15] L. Baldesi, F. Restuccia, and T. Melodia, "ChARM: NextG Spectrum Sharing Through Data-Driven Real-Time O-RAN Dynamic Control," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, May 2022.

[16] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "DeepCog: Optimizing Resource Provisioning in Network Slicing with AI-based Capacity Forecasting," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 361–376, 2019.

[17] U. Paul, J. Liu, S. Troia, O. Falowo, and G. Maier, "Traffic-profile and Machine Learning Based Regional Data Center Design and Operation for 5G Network," *Journal of Comm. and Networks*, vol. 21, no. 6, 2019.

[18] M. Polese, F. Restuccia, and T. Melodia, "DeepBeam: Deep Waveform Learning for Coordination-Free Beam Management in mmWave Networks," *Proc. of ACM MobiHoc*, 2021.

[19] A. Klautau, P. Batista, N. González-Prelcic, Y. Wang, and R. W. Heath, "5G MIMO Data for Machine Learning: Application to Beam-selection Using Deep Learning," in *Proc. of ITA Workshop*, 2018.

[20] C. Luo, J. Ji, Q. Wang, X. Chen, and P. Li, "Channel State Information Prediction for 5G Wireless Communications: A Deep Learning Approach," *IEEE Trans. on Network Science and Engineering*, vol. 7, no. 1, pp. 227–236, 2018.

[21] R. Joda, T. Pamuklu, P. E. Iturria-Rivera, and M. Erol-Kantarci, "Deep reinforcement learning-based joint user association and cu-du placement in o-ran," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2022.

[22] F. Z. Morais, G. M. de Almeida, L. Pinto, K. V. Cardoso, L. M. Contreras, R. d. R. Righi, and C. B. Both, "PlaceRAN: Optimal Placement of Virtualized Network Functions in the Next-generation Radio Access Networks," *arXiv:2102.13192 [cs.NI]*, 2021.

[23] J. M. DeAlmeida, L. DaSilva, C. B. Bonato Both, C. G. Ralha, and M. A. Marotta, "Artificial Intelligence-Driven Fog Radio Access Networks: Integrating Decision Making Considering Different Time Granularities," *IEEE Vehicular Technology Magazine*, 2021.

[24] T. S. Salem, G. Castellano, G. Neglia, F. Pianese, and A. Araldo, "Towards Inference Delivery Networks: Distributing Machine Learning with Optimality Guarantees," in *Proc. of IEEE MedHocNet*, 2021.

[25] S. D'Oro, M. Polese, L. Bonati, H. Cheng, and T. Melodia, "dApps: Distributed Applications for Real-time Inference and Control in O-RAN," *IEEE Communications Magazine*, vol. 60, no. 11, pp. 52–58, 2022.

[26] F. A. Salaht, F. Desprez, and A. Lebre, "An overview of service placement problem in fog and edge computing," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–35, 2020.

[27] Z. Usmani and S. Singh, "A survey of virtual machine placement techniques in a cloud data center," *Procedia Computer Science*, vol. 78, pp. 491–498, 2016.

[28] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "ColO-RAN: Developing Machine Learning-based xApps for Open RAN Closed-loop Control on Programmable Experimental Platforms," *arXiv:2112.09559 [cs.NI]*, 2021.

[29] L. Bonati, P. Johari, M. Polese, S. D'Oro, S. Mohanti, M. Tehrani-Moayyed, D. Villa, S. Shrivastava, C. Tassie, K. Yoder, A. Bagga, P. Patel, V. Petkov, M. Seltser, F. Restuccia, A. Gosain, K. R. Chowdhury, S. Basagni, and T. Melodia, "Colosseum: Large-Scale Wireless Experimentation Through Hardware-in-the-Loop Network Emulation," in *Proc. of IEEE DySPAN*, 2021.

[30] S. D'Oro, L. Bonati, M. Polese, and T. Melodia, "OrchestRAN: Network Automation through Orchestrated Intelligence in the Open RAN," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, May 2022.

[31] M. Polese, R. Jana, V. Kounev, K. Zhang, S. Deb, and M. Zorzi, "Machine Learning at the Edge: A Data-driven Architecture with Applications to 5G Cellular Networks," *IEEE Trans. on Mobile Computing*, 2020.

[32] Y. Li, B. Liang, and A. Tizghadam, "Robust Online Learning against Malicious Manipulation and Feedback Delay with Application to Network Flow Classification," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 8, pp. 2648–2663, 2021.

[33] T. N. Weerasinghe, I. A. Balapuwaduge, and F. Y. Li, "Supervised Learning Based Arrival Prediction and Dynamic Preamble Allocation for Bursty Traffic," in *Proc. of IEEE INFOCOM Workshops*, 2019.

[34] L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "SCOPE: An Open and Softwarized Prototyping Platform for NextG Systems," in *Proc. of ACM MobySys*, 2021.

[35] H. Chergui and C. Verikoukis, "OPEX-Limited 5G RAN Slicing: An Over-Dataset Constrained Deep Learning Approach," in *Proc. of IEEE ICC*, 2020.

[36] H. Lee, J. Cha, D. Kwon, M. Jeong, and I. Park, "Hosting AI/ML Workflows on O-RAN RIC Platform," in *Proc. of IEEE GLOBECOM Workshops*, 2020.

[37] J. A. Ayala-Romero, A. Garcia-Saavedra, X. Costa-Perez, and G. Iosifidis, "Bayesian Online Learning for Energy-Aware Resource Orchestration in Virtualized RANs," in *Proc. of IEEE INFOCOM*, 2021.

[38] R. Singh, C. Hasan, X. Foukas, M. Fiore, M. K. Marina, and Y. Wang, "Energy-Efficient Orchestration of Metro-Scale 5G Radio Access Networks," in *Proc. of IEEE INFOCOM*, 2021.

[39] S. Chatterjee, M. J. Abdel-Rahman, and A. B. MacKenzie, "On Optimal Orchestration of Virtualized Cellular Networks with Statistical Multiplexing," *IEEE Trans. on Wireless Communications*, 2021.

[40] C. Puligheddu, J. Ashdown, C. F. Chiasserini, and F. Restuccia, "SEM-O-RAN: Semantic and Flexible O-RAN Slicing for NextG Edge-Assisted Mobile Systems," in *Proceedings of IEEE INFOCOM 2023 (preprint available as arXiv:2212.11853 [cs.NI])*, New York Area, NJ, USA, May 2023.

[41] S. Matoussi, I. Fajjari, S. Costanzo, N. Aitsaadi, and R. Langar, "5G RAN: Functional Split Orchestration Optimization," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 7, pp. 1448–1463, 2020.

[42] J. Baranda, J. Mangues-Bafalluy, E. Zeydan, L. Vettori, R. Martínez, X. Li, A. Garcia-Saavedra, C.-F. Chiasserini, C. Casetti *et al.*, "On the Integration of AI/ML-based Scaling Operations in the 5Growth Platform," in *Proc. of IEEE NFV-SDN*, 2020.

[43] J. Baranda, J. Mangues-Bafalluy, E. Zeydan, C. Casetti, C. F. Chiasserini, M. Malinverno, C. Puligheddu, M. Groshev *et al.*, "Demo: AIML-as-a-Service for SLA management of a Digital Twin Virtual Network Service," in *Proc. of IEEE INFOCOM Workshops*, 2021.

[44] X. Li *et al.*, "5Growth: An End-to-End Service Platform for Automated Deployment and Management of Vertical Services over 5G Networks," *IEEE Communications Magazine*, vol. 59, no. 3, 2021.

[45] O-RAN Working Group 4, "O-RAN Fronthaul Control, User and Synchronization Plane Specification 7.0," ORAN-WG4.CUS.0-v07.00 Technical Specification, July 2021.

[46] O-RAN WG3, "O-RAN Near-Real-time RAN Intelligent Controller E2 Service Model 1.0," Technical Specification, February 2020.

[47] S. D'Oro, L. Bonati, F. Restuccia, and T. Melodia, "Coordinated 5G Network Slicing: How Constructive Interference Can Boost Network Throughput," *IEEE/ACM Trans. on Networking*, vol. 29,

Authorized licensed use limited to: Northeastern University. Downloaded on January 29,2024 at 14:47:29 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3342711

no. 4, 2021.
[48] S. D'Oro, F. Restuccia, and T. Melodia, "Toward Operator-to-Waveform 5G Radio Access Network Slicing," *IEEE Communications Magazine*, vol. 58, no. 4, pp. 18–23, April 2020.
[49] S. D'Oro, L. Bonati, F. Restuccia, M. Polese, M. Zorzi, and T. Melodia, "Sl-EDGE: Network Slicing at the Edge," in *Proc. of ACM Mobihoc*, 2020.
[50] O-RAN WG2, "O-RAN AI/ML Workflow Description and Requirements - v1.01," Technical Specification, Apr. 2020.
[51] R. Raman and I. Grossmann, "Modelling and Computational Techniques for Logic Based Integer Programming," *Computers & Chemical Engineering*, vol. 18, no. 7, pp. 563–578, 1994.
[52] R. M. Karp, "Reducibility Among Combinatorial Problems," in *Complexity of Computer Computations*, 1972, pp. 85–103.
[53] L. A. Wolsey, *Integer programming*. John Wiley & Sons, 2020.
[54] X. Li, Q. Zhai, J. Zhou, and X. Guan, "A Variable Reduction Method for Large-Scale Unit Commitment," *IEEE Trans. on Power Systems*, vol. 35, no. 1, pp. 261–272, 2020.
[55] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "OpenRAN Gym: AI/ML Development, Data Collection, and Testing for O-RAN on PAWR Platforms," *Computer Networks*, pp. 1–12, November 2022.
[56] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, "srsLTE: An Open-source Platform for LTE Evolution and Experimentation," in *Proc. of ACM WiNTECH*, 2016.

Salvatore D'Oro is a Research Assistant Professor at Northeastern University. He received his Ph.D. degree from the University of Catania in 2015. Salvatore is an area editor of Elsevier Computer Communications journal and serves on the Technical Program Committee (TPC) of multiple conferences and workshops such as IEEE INFOCOM, IEEE CCNC, IEEE ICC and IFIP Networking. He is one of the contributors to OpenRAN Gym, the first open-source research platform for AI/ML applications in the Open RAN. Dr. D'Oro's research interests include optimization, artificial intelligence, security, network slicing and their applications to 5G networks and beyond, with specific focus on Open RAN systems. He is a Member of the IEEE.

Leonardo Bonati is an Associate Research Scientist at the Institute for the Wireless Internet of Things, Northeastern University, Boston, MA. He received the Ph.D. degree in Computer Engineering from Northeastern University in 2022. His main research focuses on softwarized approaches for the Open Radio Access Network (RAN) of the next generation of cellular networks, on O-RAN-managed networks, and on network automation and orchestration. He served multiple times on the technical program committee of the ACM Workshop on Wireless Network Testbeds, Experimental evaluation & Characterization, and as guest editor of the special issue of Elsevier's Computer Networks journal on Advances in Experimental Wireless Platforms and Systems.

Michele Polese is a Principal Research Scientist at the Institute for the Wireless Internet of Things, Northeastern University, Boston, since March 2020. He received his Ph.D. at the Department of Information Engineering of the University of Padova in 2020. He also was an adjunct professor and postdoctoral researcher in 2019/2020 at the University of Padova, and a part-time lecturer in Fall 2020 and 2021 at Northeastern University. During his Ph.D., he visited New York University (NYU), AT&T Labs in Bedminster, NJ, and Northeastern University. His research interests are in the analysis and development of protocols and architectures for future generations of cellular networks (5G and beyond), in particular for millimeter-wave and terahertz networks, spectrum sharing and passive/active user coexistence, open RAN development, and the performance evaluation of end-to-end, complex networks. He has contributed to O-RAN technical specifications and submitted responses to multiple FCC and NTIA notice of inquiry and requests for comments, and is a member of the Committee on Radio Frequency Allocations of the American Meteorological Society (2022-2024). He collaborates and has collaborated with several academic and industrial research partners, including AT&T, Mavenir, NVIDIA, InterDigital, NYU, University of Aalborg, King's College, and NIST. He was awarded with several best paper awards, is serving as TPC co-chair for WNS3 2021-2022, as an Associate Technical Editor for the IEEE Communications Magazine, and has organized the Open 5G Forum in Fall 2021. He is a Member of the IEEE.

Tommaso Melodia is the William Lincoln Smith Chair Professor with the Department of Electrical and Computer Engineering at Northeastern University in Boston. He is also the Founding Director of the Institute for the Wireless Internet of Things and the Director of Research for the PAWR Project Office. He received his Ph.D. in Electrical and Computer Engineering from the Georgia Institute of Technology in 2007. He is a recipient of the National Science Foundation CAREER award. Prof. Melodia has served as Associate Editor of IEEE Transactions on Wireless Communications, IEEE Transactions on Mobile Computing, Elsevier Computer Networks, among others. He has served as Technical Program Committee Chair for IEEE Infocom 2018, General Chair for IEEE SECON 2019, ACM Nanocom 2019, and ACM WUWnet 2014. Prof. Melodia is the Director of Research for the Platforms for Advanced Wireless Research (PAWR) Project Office, a $100M public-private partnership to establish 4 city-scale platforms for wireless research to advance the US wireless ecosystem in years to come. Prof. Melodia's research on modeling, optimization, and experimental evaluation of Internet-of-Things and wireless networked systems has been funded by the National Science Foundation, the Air Force Research Laboratory the Office of Naval Research, DARPA, and the Army Research Laboratory. Prof. Melodia is a Fellow of the IEEE and a Senior Member of the ACM.

Authorized licensed use limited to: Northeastern University. Downloaded on January 29,2024 at 14:47:29 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.