# O-RAN Intelligence Orchestration Framework for Quality-Driven xApp Deployment and Sharing

Federico Mungari, *Student Member, IEEE*, Corrado Puligheddu, *Member, IEEE*, Andres Garcia-Saavedra, Carla Fabiana Chiasserini, *Fellow, IEEE*

***Abstract*—The rapid evolution of 5G networks, with diverse traffic classes and demanding services, highlights the importance of Open Radio Access Networks (O-RAN) for enabling RAN intelligence and performance optimization. Machine Learning-powered xApps offer novel network control opportunities, but their resource demands necessitate efficient orchestration. To address these issues, we present OREO, an O-RAN xApp orchestrator that, using a multi-layer graph model, aims to maximize the number of RAN services concurrently deployed while minimizing their overall energy consumption. OREO's key innovation lies in the concept of sharing xApps across RAN services when they include semantically equivalent functions and meet quality requirements. Despite the NP-hard nature of the problem, numerical results show that OREO offers a lightweight and scalable solution that closely and swiftly approximates the optimum in several different scenarios. Also, OREO outperforms state-of-the-art benchmarks by enabling the co-existence of more RAN services (14.3% more on average and up to 22%), while reducing resource expenditure (by 48.7% less on average and up to 123% for computing resources). Moreover, using an experimental prototype deployed on the Colosseum network emulator and using real-world RAN services, we show that OREO leads to substantial resource savings (up to 66.7% of computing resources) while its xApp sharing policy can significantly enhance quality of service.**

***Index Terms*—Radio Access Network, Resource orchestration, O-RAN, Network services**

## I. Introduction

As 5G mobile networks continue to gain momentum, the limitations of the traditional monolithic Radio Access Network (RAN) architecture have emerged in accommodating a growing number of traffic classes and demanding services [2]. Indeed, conventional RAN architectures encompass monolithic components that are tightly bundled and supplied by a handful of vendors. This approach poses challenges such as limited reconfigurability, which hinders support for innovative applications and restricts deployment options for operators due to vendor lock-in [3].

F. Mungari, C. Puligheddu, and C. F. Chiasserini are with Politecnico di Torino, Italy and CNIT, Italy. C. F. Chiasserini is also with Chalmers University of Technology, Sweden. A. Garcia-Saavedra is with NEC Laboratory Europe, Spain.

Research reported in this publication was supported by the Qatar Research Development and Innovation Council ARG01-0525-230339. The content is solely the responsibility of the authors and does not necessarily represent the official views of Qatar Research Development and Innovation Council. The work was also supported by the European Commission through Grant No. 101095890 (PREDICT-6G project) and Grant No. 101139270 (ORIGAMI project), and by the EU under the Italian NRRP of NextGenerationEU, partnership on "Telecommunications of the Future" (PE0000001 - program "RESTART").
This is an extended version of our IEEE INFOCOM 2024 paper [1].

To overcome the rigidity of deploying and orchestrating the network and its services, embracing cutting-edge RAN solutions grounded in virtualization and openness principles is essential [4], [5]. To this end, the O-RAN Alliance spearheads the development of O-RAN, which promotes openness and flexibility in the RAN by defining open standard interfaces and ensuring compatibility between disaggregated multi-vendor components. Additionally, O-RAN integrates Radio Intelligent Controllers (RICs) within the RAN, facilitating the collection of telemetry data and the implementation of customized control logic.

O-RAN enables Mobile Network Operators (MNOs) to deliver intelligent network services within the RAN. Hereafter, we refer to these as "RAN services" or simply "services." RAN services are specialized functionalities designed to enhance the overall performance, efficiency, and adaptability of mobile networks. These services typically leverage intelligent and automated processes to optimize resource allocation, improve user experience, and support network operations. Examples include dynamic spectrum management, traffic steering, energy-efficient operation, and real-time network analytics [6], [7].

RICs facilitate the deployment of services by hosting third-party applications driven by Artificial Intelligence (AI) and Machine Learning (ML), enabling closed-loop control and self-optimization. These applications operate on diverse timescales, ranging from sub-second (*xApps*) to longer-term (*rApps*) optimization.

Rooting *xApps* in AI/ML places substantial demands on O-Cloud resources – the computing platform hosting O-RAN software components. However, the O-Cloud offers limited computing resources, shared among concurrently running *xApps* from potentially different tenants. Consequently, it becomes crucial to minimize the computational footprint of these apps while still meeting the target performance of the services they provide. Note that reducing resource consumption in the O-Cloud not only decreases the operational expenditure of RANs, representing 40% of the total costs in cellular network [8], but it also contributes to decrease energy consumption.

**Existing research gap.** Designing an efficient policy for orchestrating RAN intelligence in O-RAN platforms remains an open challenge. While numerous state-of-the-art orchestration frameworks exist (as discussed in Sec. VII), they fall short in fully addressing the complexities of O-RAN management. These frameworks often focus either on traditional RAN orchestration problems, such as optimizing resource management and energy efficiency, or partially tackle O-RAN intelligence orchestration by solely addressing deployment without optimizing configuration. Conversely, some cutting-edge frameworks offer solutions to O-RAN intelligence orchestration but consider only monolithic services and xApps with oversized resource allocations, leading to sub-optimal RAN management.

Filling this gap presents multiple technical challenges. Foremost is the development of an orchestration *policy* that can identify the optimal service configuration. This involves several key steps: ($i$) selecting the most suitable set of xApps from the near-real-time (near-RT) RIC's catalog, ensuring they align with the service's functional requirements; ($ii$) verifying that the selected xApps have the capability to effectively execute the specific functions needed for the service; ($iii$) determining the appropriate complexity level for the xApps to guarantee they deliver output with sufficient accuracy and confidence to meet the service's quality standards; and ($iv$) allocating computational resources (CPU, GPU) and memory (RAM, disk space) to the xApps, finding a balance between meeting service latency requirements and staying within the available resource budget.

**Summary of novel contributions.** To address this problem, we introduce the O-RAN intElligence Orchestration (OREO) framework. OREO determines the optimal selection and configuration of *xApps* to fulfill network service demands provided by MNO, meeting specific requirements while minimizing resource expenditure. Compared to the state-of-the-art, OREO provides the following distinct key features:

- **NFV Integration:** OREO embraces Network Function Virtualization (NFV) within O-RAN, conceiving RAN services (e.g., beam allocation, handover prediction) as interconnected elementary RAN functions (e.g., load forecasting, traffic classification, decision policies). This approach fosters service agility, flexibility, scalability, and enables the sharing of common functions across services.
- **Scope-aware Optimization:** OREO optimizes resource allocation to *xApps*, taking into account their ability to operate across various scopes (i.e., operational semantics) to exploit *xApps* sharing supporting concurrent services.
- **Function Variability:** The dynamic O-RAN market fosters diverse implementations for individual functions, resulting in different trade-offs between output quality, resource consumption, and execution speed. As detailed in Sec. II-A, OREO exploits such diversity (which we codify with different complexity levels) for the deployment of functions within *xApps* to maximize efficiency while providing service performance guarantees.

We evaluate OREO through extensive numerical as well as experimental results obtained through the Colosseum network emulator [9]. Our results show that OREO can support a number of services close to the optimum, and, compared to state-of-the-art solutions, OREO enables the co-existence of more services (14.3% more on average and up to 22%), while reducing resource expenditure (by 48.7% less on average and up to 123% for computing resources). Moreover, using our experimental prototype deployed on the Colosseum emulator and using real-world RAN services, we show that OREO leads to substantial resource savings (up to 66.7% of computing resources) while its xApp sharing policy can significantly enhance quality of service (up to 11.3% increase in average throughput and 13.1% reduction in buffer occupancy for, respectively, Enhanced Mobile Broadband (eMBB) and Ultra-Reliable Low Latency Communication (URLLC) users).

This paper also extends substantially our previous conference publication [1]:

- We have re-designed a large portion of OREO's framework to enhance agility. We have achieved this through a more efficient decomposition approach and pruning technique to reduce the solution space.
- We have extended our numerical evaluation, which now shows OREO's ability to orchestrate RAN services even in very large scenarios.
- We have improved the experimental assessments of OREO by conducting tests using Colosseum [9], the well-known O-RAN emulator, and real-world RAN services. These new experiments demonstrate OREO's effectiveness in resource allocation and its ability to meet service KPI targets.

**Paper organization.** The rest of the paper is organized as follows. Sec. II introduces the driving purpose and distinctive features of OREO framework. Sec. III presents the system model and formulates the xApp Deployment and Sharing (xDeSh) problem. The iterative heuristic method we designed to address the xDeSh problem is described in Sec. IV. The latter involves Lagrangian relaxation and decoupling methods (Sec. IV-B), followed by an algorithm ensuring the feasibility of the obtained solution (Sec. IV-D). Sec. V and Sec. VI present, respectively, the numerical evaluation and the experimental prototype and validation of OREO. Sec. VII discusses some relevant work in the RAN intelligence orchestration field. Finally, Sec. VIII concludes the paper.

## II. The OREO Framework

This section presents the OREO framework, first outlining its purpose in an O-RAN system and describing the distinctive features of its engine (Sec. II-A), and then providing the rationale behind its design and its integration within the O-RAN architecture (Sec. II-B).

### A. OREO driving purpose and distinctive features

The success of next-generation mobile networks greatly hinges upon the quality of RAN intelligence and, hence, upon the performance of its orchestration framework, which is responsible for deploying RAN services [10].

Our orchestrator, OREO, acts upon a set of service requests by the MNOs in an O-RAN platform. Given such requests, OREO selects the xApp(s) required to deploy the services in the near-RT RIC, and the specific xApps configuration that lets a service meet its performance requirements while matching the resource availability in the platform. OREO's orchestration decisions are made by its core component, the OREO engine, which greatly differs from state-of-the-art O-RAN orchestrators such as the pioneering work in [11].

A key differentiating principle that drives the design of the OREO engine consists of conceiving RAN services as sets of interconnected functions rather than monolithic entities. This approach capitalizes on the known benefits of the NFV paradigm, such as enhanced flexibility, scalability, and cost-effectiveness [12]. By recognizing that RAN services can be built by interconnecting elementary RAN management functions, OREO leverages xApps as fundamental building blocks to efficiently offer such services. More specifically,

- *Service composition:* Each network service request fed by an MNO to the OREO engine is associated with a minimum service quality and a maximum response latency requirement. Depending on such performance targets, services can be deployed by using different configurations: each configuration corresponds to a different set of functions, with each function implementing a certain task. As an example, network slicing can be enabled through a single function implementing a reinforcement learning (RL)-based policy, as in [13], or combining such function with a traffic predictor that feeds its output to the RL model for improved system response to changes in the traffic conditions.
- *Implementing service functions through xApps:* Additionally, each function can be implemented through multiple xApps, each instantiated at a different operating point, hereinafter also referred to as *complexity factor*. Importantly, complexity factors provide different trade-offs between the output quality and processing latency of the function offered by the xApp and the computational resources necessary to run that xApp.

Thus, based on the above concepts, in OREO the quality and response latency incurred by a service depend upon:

- The specific configuration (set of functions) that is selected to enable the service;
- The specific xApps (hence levels of complexity) that are chosen to implement the functions in the selected configuration.

The OREO engine identifies the service configuration *and* the corresponding xApps in such a way that it can best suit the service requirements. Furthermore,

- Whenever multiple services require the same function, *OREO allows such services to share the xApp that implements the semantics of that function*, if the xApp complexity level meets the requirements of the services;
- As the service response latency targets can be fulfilled by properly setting the resources allocated to the shared xApps, *OREO scales the resources assigned to an xApp according to the overall load imposed by the corresponding services*, as well as the available resource budget. Importantly, in so doing, OREO avoids resource over-provisioning, as opposed to relying upon a fixed amount of resources allocated to xApps as in state-of-the-art solutions [11].

## B. OREO system architecture

The OREO framework, illustrated in Fig. 1, is designed to be integrated into the O-RAN Service Management and Orchestration (SMO), which is responsible for managing and orchestrating all control and monitoring procedures of the RAN components via the O1 interface. In particular, OREO operates within the non-RT RIC, which supports the execution of third-party applications known as rApps and, through the A1 interface and the support of the Near-RT RIC, enables closed-loop control of the RAN. The entire O-RAN deployment runs in the O-Cloud, a computing platform comprising a collection of physical infrastructure nodes that meet O-RAN requirements to host the relevant O-RAN Network Functions (NFs) (i.e., Near-RT RIC, O-CU-CP, O-CU-UP, and O-DU), the supporting software components (such as Operating System, Virtual Machine Monitor, Container Runtime, etc.), and the appropriate management and orchestration functions [14].

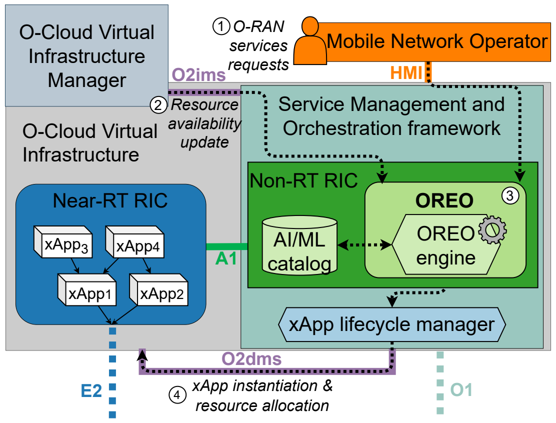
O-Cloud Virtual Infrastructure Manager
① O-RAN services requests
Mobile Network Operator
O2ims
HMI
② Resource availability update
Service Management and Orchestration framework
O-Cloud Virtual Infrastructure
Non-RT RIC
Near-RT RIC
OREO ③
xApp3
xApp4
A1
AI/ML catalog
OREO engine
xApp1
xApp2
xApp lifecycle manager
O2dms
E2
④ xApp instantiation & resource allocation
O1

Fig. 1: OREO design and integration in the O-RAN architecture. The workflow (dashed black line) is as follows: ($i$) the MNO submits service requests via the HMI; ($ii$) OREO retrieves the updated status of computing resources from the O-Cloud Virtualized Infrastructure Manager via the O2ims interface; ($iii$) OREO processes service requests and, with the support of the xApp lifecycle manager, instructs the O-Cloud via the O2dms interface about which xApps to deploy in the near-RT RIC .

OREO receives RAN management intents from MNOs via the Human-Machine Interface (HMI) [15]. These intents define the requested services and outline specific parameters, including the maximum acceptable delay and minimum quality requirements for each service (Step 1). OREO fetches the O-Cloud resource availability from Virtual Infrastructure Manager through the O2 Infrastructure Management Service (O2ims) interface (Step 2). As mentioned in the previous section, the OREO engine calculates the deployment of xApps satisfying the service requests and resource availability (Step 3). xApps are indeed third-party applications that implement customized logic to drive the RAN efficiently and run within the near-RT RIC, i.e., the central control and optimization unit of the RAN operating on a sub-second time scale. The selected xApps are deployed within the near-RT RIC using management services, such as the xApp lifecycle manager provided by the SMO through the O2 Deployment Management Service (O2dms) interface (Step 4). Through the E2 interface and open APIs, the near-RT RIC interacts with the RAN centralized and distributed units (O-CUs and O-DUs, respectively), collecting RAN performance metrics and providing control actions.

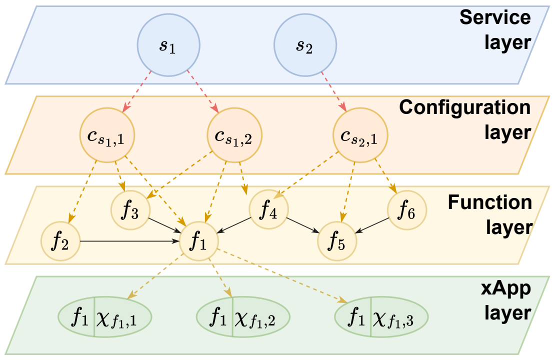

Fig. 2: Graph-based representation of the system under study and relation between its main components. A RAN service can be offered through many configurations, i.e., combinations of dependent functions. Functions are abstract elementary blocks defining an operation to perform but not how to perform it. The concrete implementation of a function, i.e., an xApp, comprises a complexity level indicating how accurate and resource-demanding it is. For clarity, the xApp layer only includes the xApps implementing function $f_1$.

As detailed in Sec. VI, we have implemented all OREO components and integrated them in the O-RAN architecture, leveraging an O-Cloud platform [16] that hosts the SMO, the non-RT RIC, and the near-RT RIC. In doing so, we have developed a proof-of-concept testbed and, using that, we have measured the performance of the proposed solution in real-world settings.

## III. xDeSh: xApp Deployment and Sharing

We now introduce a model that captures all relevant system aspects (Sec. III-A) and formulate the xApp Deployment and Sharing (xDeSh) problem (Sec. III-B).

### A. System model

Fig. 2 depicts the main system components, which we further detail in the following; the notation used in this section is summarized in Table I.

- **Services.** An O-RAN service refers to any autonomous network control and performance optimization service, along with its associated tasks, operating within the RICs. In particular, we focus on decision-making services managed by the near-RT RIC. The O-RAN Alliance has identified several services [6], [7], including among others, traffic steering (e.g., [17]), handover management (e.g., [18]), and QoS-based resource optimization (e.g., [19]). Each service $s$ is characterized by: $(i)$ a target response latency, $T_s$, which specifies the maximum acceptable delay to output a decision since a service request arrives; and $(ii)$ a target decision quality, $Q_s$. The specific interpretation of *quality* depends on the nature of the service (e.g., accuracy for a traffic classification service) but we assume that services can be objectively assessed. Further, a service is assigned a priority level $p_s$, which depends on the revenue generated for the MNO and is used to determine which services should be dropped in case of insufficient resource availability.
- **Service configurations.** A service can be provided using different configurations, $c_s \in \mathcal{C}_s$, i.e., sets of interconnected elementary functions. Each service configuration is associated with a level of quality and resource demand, determined by the set of functions appearing in the configuration. Following the NFV practices [20], we assume that MNOs pre-define, often manually, and input the catalog of available service configurations. Thus, by properly selecting $c_s$ makes it possible to trade off the performance of a service with its deployment and running cost.

TABLE I: Notations

| **Parameters** | |
|---|---|
| **Symbol** | **Description** |
| $s{\in}\mathcal{S}$ | RAN service |
| $T_s$ $(Q_s)$ | Target latency of service $s$ under configuration $c_s$ |
| $p_s$ | Priority of service $s$ |
| $c_s{\in}\mathcal{C}_s$ | Service $s$ configuration |
| $\mathcal{V}_{c_s}$ | Set of nodes of the service configuration graph $c_s$ |
| $f{\in}\mathcal{F}$ | RAN function |
| $\chi_f{\in}\mathcal{X}_f$ | Complexity factor of function $f$ |
| $f_\chi$ | xApp implementing function $f$ with complexity $\chi$ |
| $f_\chi^{(j)}$ | $j$-th instance of xApp $f_\chi$ |
| $\mu_{f_\chi^{(j)},\text{mem (disk)}}$ | Memory (disk) requirement of xApp $f_\chi^{(j)}$ |
| $\lambda_{\mathcal{P}(f_\chi^{(j)})}$ | Input data rate of $f_\chi^{(j)}$ if shared among $s{\in}\mathcal{P}(f_\chi^{(j)})$ |
| $\theta_{f_\chi}$ | Amount of input data processed by $f_\chi$ in a CPU cycle |
| $\mathcal{K}$ | Set of resource types |
| $\mathbf{B}$ | Vector of resource budgets of the different types |
| $q_{c_s,f_\chi}$ | Quality of xApp $f_\chi$ |
| $l_{f_\chi^{(j)}}$ | Processing latency of the j-th instance of xApp $f_\chi$ |
| $\tau_{c_s}$ | Latency of service $s$ |
| **Decision variables** | |
| **Symbol** | **Description** |
| $z_{c_s}$ | Binary variable for service configuration $c_s$ selection |
| $v_{c_s,f_\chi^{(j)}}$ | Binary variable indicating if $f_\chi^{(j)}$ is used in configuration $c_s$ |
| $\boldsymbol{\rho}_{f_\chi^{(j)}}$ | Resource allocation for the j-th instance of xApp $f_\chi$ |

- **Functions.** A function $f{\in}\mathcal{F}$ represents a low-level operation and serves as the fundamental building block of one or more services. Examples of functions include traffic forecasting and traffic classification. Functions may process $(i)$ metrics collected by the RAN elements (O-DU, O-CU, etc.) and shared with the near-RT RIC via the E2 interface; and/or $(ii)$ information provided by other functions. A service configuration can then be modeled as a directed graph whose vertices ($\mathcal{V}_{c_s}$) and edges represent, respectively, the functions composing the configuration and the dependency relations between them. Specifically, an edge exists from function $f'$ to function $f$ whenever the execution of $f$ requires the output of $f'$.
- **xApps.** Each function can be implemented with a different *complexity factor*, $\chi_f{\in}\mathcal{X}_f$. For instance, a traffic classification function can be provided by different ML models, each offering a different accuracy-resource demand trade-off. A specific function with a given complexity factor defines an xApp, which is thus indicated as $f_\chi = (f, \chi_f)$. The xApps may require specialized data, potentially pre-processed. In this work, we assume that xApps are designed to handle the necessary data pre-processing, and that their input/output interfaces are fully compatible with the O-RAN platform. Furthermore, as exemplified in Fig. 3, xApps can be shared by different service configurations. Let $\mathcal{P}(f_\chi)$ be the set of

service configurations that include xApp $f_\chi$, and $\lambda_{\mathcal{P}(f_\chi)}$ be the rate at which data is fed to, and needs to be processed by, the xApp per unit of time. We remark that multiple instances (i.e., replicas) of a given xApp can be implemented and, hence, coexist in the system; we then denote the $j$-th instance of $f_\chi$ with $f_\chi^{(j)}$.

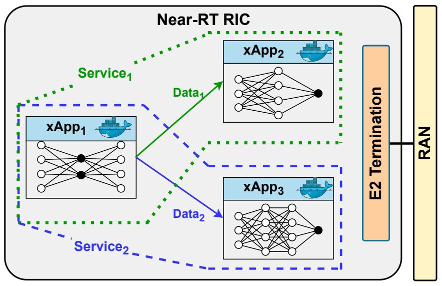

Fig. 3: Illustration of xApp sharing between two services, which are identified through the green and blue dashed polygons. In this example, xApp$_1$ operates across two distinct domains, which may overlap if both services require the xApp with the same operational semantics.

- **Near-RT RIC resources.** The O-Cloud can provide the near-RT RIC with computing and storage resources (e.g., CPU, GPU, memory) to run xApps. We denote with $\mathcal{K}=\{1,\ldots,K\}$ the set of available resource types, and we differentiate among computing resources as $\mathcal{K}_\tau$ and storage resources as $\mathcal{K}_q$. The vector $\boldsymbol{B}=[B_1,\ldots,B_K]$ collects, for each resource type, the available resource budget. For simplicity and without loss of generality, in the following we focus on CPU within $\mathcal{K}_\tau$, and memory and disk storage within $\mathcal{K}_q$. Thus, we let $\boldsymbol{\rho}_{f_\chi^{(j)}}=[\rho_{f_\chi^{(j)},1},\ldots,\rho_{f_\chi^{(j)},K}]$, with $\rho_{f_\chi^{(j)},k}\le\boldsymbol{B_k}$ denote the amount of resource of type $k$ reserved for the xApp instance $f_\chi^{(j)}$, and the corresponding memory and disk requirements with, respectively, $\mu_{f_\chi^{(j)},\text{mem}}$ and $\mu_{f_\chi^{(j)},\text{disk}}$.
- **xApp and service quality.** Let $q_{c_s,f_\chi}$ be the quality score obtained by the xApp $f_\chi$ associated with service configuration $c_s$. This score serves as a numerical metric that assesses the performance of the xApp within its designated service. The precise meaning of this metric can vary based on the particular focus of the xApp. For example, in traffic classification tasks, the quality score typically reflects classification accuracy. Other common quality metrics might include prediction accuracy, regression error, or expected reward. The quality score $q_{c_s,f_\chi}$ depends on the quality of the input data, which, in turn, depends on the complexity level associated with the function $f'$ preceding $f$ in the configuration graph. Accordingly, the quality metric for a service $s$ implemented under configuration $c_s$, is equal to the quality of the last xApp's output in the configuration graph.
- **xApp processing and service latency.** Given the considered resource types, as long as memory and disk requirements ($\mu_{f_\chi^{(j)},\text{mem}}$ and $\mu_{f_\chi^{(j)},\text{disk}}$) are satisfied, only the CPU allocation has an impact on the xApps processing latency, denoted by $l_{f_\chi^{(j)}}(\rho_{f_\chi^{(j)},\text{cpu}})$ for the $j$-th instance of xApp $f_\chi$. Drawing on the existing works [21]–[23], we can model a function, $f_\chi^{(j)}$, that is shared among the service configurations in $\mathcal{P}(f_\chi^{(j)})$, as an M/M/1 queue. We can then write the corresponding average processing latency as:

$$l_{f_\chi^{(j)}}(\rho_{f_\chi^{(j)},\text{cpu}})=(\rho_{f_\chi^{(j)},\text{cpu}}\theta_{f_\chi^{(j)}}-\lambda_{\mathcal{P}(f_\chi^{(j)})})^{-1}$$

where $\rho_{f_\chi^{(j)},\text{cpu}}$ is expressed as CPU cycles per second, and $\theta_{f_\chi^{(j)}}$ represents the xApp complexity and expresses the amount of input data processed by the xApp in a CPU cycle, and $\lambda_{\mathcal{P}(f_\chi^{(j)})}$ specifies the xApp load when shared among $\mathcal{P}(f_\chi^{(j)})$ service configurations. The latter must account for all the operational semantics (i.e., scopes), required by the xApp when it is shared across services. For instance, a traffic predictor depending on the specific network service can function across various domains – such as user, slice, or cell – or operate across multiple gNodeBs. Also, let $\tau_{c_s}$ be the response latency of service $s$ when implemented with configuration $c_s$. By defining a path $\pi_{c_s}$ on the graph of configuration $c_s$ as a set of edges connecting an input function with an output function in $c_s$, $\tau_s$ is the latency associated with the most time-consuming path in the graph. The latency for collecting data is indeed deemed negligible as the near-RT RIC periodically exposes data to the xApps. The latency of a path clearly depends on the complexity factor and resource allocation of each of the functions composing the path, i.e.,

$$\tau_{c_s}(\{\boldsymbol{\rho}_{f_\chi},\chi_f\}_{f_\chi\in c_s})=\arg\max_{\{\pi_{c_s}\}}\sum_{f\in\pi_{c_s}}l_{f_{j,\chi_{f_j}}}(\boldsymbol{\rho}_{f_j})\,. \tag{1}$$

## B. xDeSh problem formulation

Given the above model, we now introduce the xDeSh optimization problem, along with some additional system variables and parameters defining the current state of the system. Further, we prove that the xDeSh problem is NP-hard.

- **Service configuration selection.** Let $\mathcal{S}$ be the set including both the existing, and still to be kept, services and the new services to be deployed. For each service $s\in\mathcal{S}$, the OREO orchestrator identifies the most suitable configuration $c_s$ to be used. We denote with $z_{c_s}$ the binary decision variable taking 1 if configuration $c_s$ is selected for service $s$. Notice that: $(i)$ it may happen that none of the possible configurations of a service $s$ can be deployed, due to insufficient resources to guarantee the minimum required service performance; $(ii)$ at most one configuration per service can be selected. That is, the following constraint must hold:

$$\sum_{c_s\in\mathcal{C}_s}z_{c_s}\le 1,\,\forall s\in\mathcal{S}\,. \tag{2}$$

- **Selection of xApps to implement and share.** Whenever a function $f$ is required by more than one service, the orchestrator has to determine whether to let such services share *the same xApp* instance $f_\chi^{(j)}$, or to implement multiple instances thereof. In the latter case, the chosen xApps can implement either the same or different complexity factors. We thus introduce the binary decision variable $v_{c_s,f_\chi^{(j)}}$, to indicate whether $f_\chi^{(j)}$ is used by service configuration $c_s$ or not. Clearly, all the functions required by a selected service configuration must be implemented. Moreover, neglecting the possibility of horizontally scaling out xApps, a service configuration cannot use more than one instance of a given xApp. The above requirements translate in the following constraint:

$$\sum_{\chi\in\mathcal{X}_f}\sum_{j} v_{c_s,f_\chi^{(j)}} = z_{c_s},\ \forall s\in\mathcal{S},\forall c_s\in\mathcal{C}_s,\forall f\in\mathcal{V}_{c_s}\,. \quad (3)$$

Similarly, an xApp implementing function $f$ cannot be associated with a service configuration that does not include $f$:

$$\sum_{\chi\in\mathcal{X}_f}\sum_{j} v_{c_s,f_\chi^{(j)}} = 0,\ \forall s\in\mathcal{S},c_s\in\mathcal{C}_s,f\notin\mathcal{V}_{c_s}\,. \quad (4)$$

Ultimately, the orchestrator allocates memory and disk resources for deploying the necessary xApps, while adhering to the following constraint:

$$\rho_{f_\chi^{(j)},k}\geq\mu_{f_\chi^{(j)},k}\mathbb{1}_{[\exists c_s\in\mathcal{C}_s|v_{c_s,f_\chi^{(j)}}=1]},\ \forall k\in\mathcal{K}_q \quad (5)$$

where the indicator function $\mathbb{1}$ equals one when the subscripted condition holds, indicating that xApp $f_\chi^{(j)}$ must be deployed.

• **Meeting service requirements.** OREO has to select a service configuration (i.e., the functions that implement the service) and vertically scales the corresponding computing resources in such a way that the quality and response latency targets are satisfied. That is, for any $s\in\mathcal{S}$ and $c_s\in\mathcal{C}_s$,

$$q_{c_s}(\{\chi_f \mid \sum_j v_{c_s,f_\chi^{(j)}} = 1\}_{f\in c_s}) \geq Q_s\cdot z_{c_s} \quad (6)$$

$$\tau_{c_s}(\{\boldsymbol{\rho}_{f_\chi^{(j)}},\chi_f \mid \sum_j v_{c_s,f_{\chi_f}^{(j)}} = 1\}_{f\in c_s})\cdot z_{c_s}\leq T_s\,. \quad (7)$$

• **Complying with the resource budget.** We also need conventional capacity constraints, i.e., the near-RT RIC resource budget $\boldsymbol{B}$ must not be exceeded:

$$\sum_{f\in\mathcal{F}}\sum_{\chi\in\mathcal{X}}\sum_{j}\rho_{f_\chi^{(j)},k}\leq B_k,\ \forall k\in\mathcal{K}\,. \quad (8)$$

• **Avoid service disruption.** It is critical to account for the cost incurred by the system whenever OREO determines a new configuration for an existing service $s$. Let $\hat{\mathcal{S}}=\{\hat{s}\}$ denote the set of services that are already deployed, and $c_{\hat{s}}$ capture their service configuration. Accordingly, the binary parameter $\hat{z}_{c_{\hat{s}}}$ takes 1 if configuration $c_{\hat{s}}$ of service $\hat{s}$ is implemented, and 0 otherwise. Now, given an xApp instance $f_\chi^{(j)}$, we let $\hat{v}_{c_{\hat{s}},f_\chi^{(j)}}$ indicate whether service configuration $c_{\hat{s}}$ is using $f_\chi^{(j)}$, and $\hat{\boldsymbol{\rho}}_{f_\chi^{(j)}}$ denote the vector indicating its current resource allocation.

To ensure continuity for a service $s\in\hat{\mathcal{S}}\cap\mathcal{S}$, both the xApps required by $\{c_{\hat{s}}\mid\hat{z}_{c_{\hat{s}}}\}_{\hat{s}\in\hat{\mathcal{S}}\cap\mathcal{S}}$ and by $\{c_s\mid z_{c_s}\}_{s\in\hat{\mathcal{S}}\cap\mathcal{S}}$ have to coexist before $(i)$ turning off the relative currently implemented, but no longer required, xApps, and $(ii)$ instantiating the remaining functions required by the residual services in $\mathcal{S}$. We then define $\mathcal{F}_1$ as the set of xApps required by the existing services that should not be deactivated:

$$\mathcal{F}_1=\{f_\chi^{(j)}\mid\sum_{c_{\hat{s}}\in\mathcal{C}_{\hat{s}}}\hat{v}_{c_{\hat{s}},f_\chi^{(j)}}=1\}_{f\in\mathcal{F},\,\chi_f\in\mathcal{X}_f,\,j,\,\hat{s}\in\hat{\mathcal{S}}\cap\mathcal{S}}\,.$$

Similarly, let $\mathcal{F}_2$ be the set of xApps required in the last defined near-RT RIC's setting for the services whose operation must not be disrupted. Then, to avoid service disruptions, we must have:

$$\sum_{f_\chi^{(j)}\in\mathcal{F}_1\setminus\mathcal{F}_2}\hat{\rho}_{f_\chi^{(j)},k}+\sum_{f_\chi^{(j)}\in\mathcal{F}_2}\rho_{f_\chi^{(j)},k}\leq B_k,\ \forall k\in\mathcal{K} \quad (9)$$

where $\mathcal{F}_1\setminus\mathcal{F}_2=\{f_\chi^{(j)}\mid f_\chi^{(j)}\in\mathcal{F}_1\wedge f_\chi^{(j)}\notin\mathcal{F}_2\}$.

• **Objective function.** The xDeSh problem aims to define an xApp selection and resource allocation policy that $(i)$ maximizes the number of offered services based on their priority levels, and $(ii)$ minimizes the near-RT RIC resource consumption. Accordingly, we describe the problem objective function as:

$$\Psi(\boldsymbol{z},\boldsymbol{v},\boldsymbol{\rho})=\sum_{s\in\mathcal{S}}\sum_{c_s\in\mathcal{C}_s}z_{c_s}\,p_s-\frac{1}{K}\sum_{f\in\mathcal{F}}\sum_{\chi\in\mathcal{X}_f}\sum_{j}\sum_{k\leq K}\frac{\rho_{f_\chi^{(j)},k}}{B_k}$$

where the decision variables (see Table I) have been vectorized, and the $1/K$ factor prevents service rejection for the sake of resource savings. The xDeSh problem can then be formulated as:

**xApp Deployment and Sharing (xDeSh) Problem**

$$\max_{\boldsymbol{z},\boldsymbol{v},\boldsymbol{\rho}}\ \Psi(\boldsymbol{z},\boldsymbol{v},\boldsymbol{\rho})$$
$$\text{s.t.}\ (2),(3),(4),(5),(6),(7),(8),(9)$$
$$z_{c_s}\in\{0,1\}\ \forall s\in\mathcal{S},c_s\in\mathcal{C}_s$$
$$v_{c_s,f_{\chi_f}^{(j)}}\in\{0,1\}\ \forall c_s\in\mathcal{C}_s,f\in\mathcal{F},\chi_f\in\mathcal{X}_f,j$$
$$\rho_{f_\chi^{(j)},k}\in[0,B_k]\ \forall k\in\mathcal{K},f\in\mathcal{F},\chi_f\in\mathcal{X}_f,j$$

**Proposition 1.** *The xDeSh problem is NP-hard.*

*Proof.* We show that any instance of the well-known NP-hard multi-commodity facility location problem (FLP) can be reduced to an instance of the xDeSh problem. To this end, we first recall that FLP aims to determine $(i)$ the optimal location for deploying facilities, and $(ii)$ the commodities that each facility offers to fulfill the consumers' requests while minimizing construction costs [24]. We then focus on a simplified version of the xDeSh problem where each service can have multiple configurations, but each configuration consists of a single function ($|\{f\}_{f\in\mathcal{C}_s}|=1,\ \forall s\in\mathcal{S},c_s\in\mathcal{C}_s$). We disregard the ability to activate functions with different complexity factors ($|\mathcal{X}_f|=1,\ \forall f\in\mathcal{F}$) and assume that the minimum resource allocation for functions to meet the target service performance is known. Also, we neglect reconfiguration costs.

The following mapping can then be defined between the entities in the multi-commodity FLP and the reduced xDeSh problem: $(i)$ facility deployment locations correspond to functions; $(ii)$ commodities available at facilities correspond to the services that functions can provide; $(iii)$ customers are the MNOs; $(iv)$ the construction cost corresponds to the function deployment cost. By further observing that the above reduction can be obtained in polynomial time, the thesis follows. ■

## IV. Solving the xDeSh Problem

Motivated by Proposition 1 above, we propose an efficient iterative heuristic to solve the problem in Sec. III-B. We outline such solution in Sec. IV-A, and then detail each of its building blocks, namely,

(i) a Lagrangian relaxation of the original problem and the decoupling method we adopt to reduce its complexity (Sec. IV-B);
(ii) the pruning technique we apply to reduce the solution space (i.e., further reduce the solution complexity) (Sec. IV-C);
(iii) an algorithm to ensure that the obtained solution is feasible (Sec. IV-D), and, finally,
(iv) a subgradient method for updating the Langrange multipliers (Sec. IV-E).

As we introduce each stage of the heuristic solution, we also analyze its computational complexity (Sec. IV-F). We remark that our solution algorithm is executed in the OREO engine, introduced in Sec. II.

### A. Overview of the algorithmic solution

In the proof of Prop. 1, we underlined the similarity between the xDeSh problem and the FLP. Inspired by existing efficient FLP solvers [25]–[27], we design our algorithmic solution adopting an iterative, two-stage approach. As illustrated in Fig. 4, our solution framework first leverages the Lagrangian Relaxation (LR) method, a relaxation technique that incorporates the effect of the constraints that entail the problem's complexity into the objective function. To enforce these constraints, the method introduces penalty terms, i.e., Lagrange multipliers. However, this approach may provide a solution to the xDeSh problem that is infeasible.

To solve this issue, we combine the LR method with an algorithm capable of identifying the violated constraints and making adjustments to the relaxed solution. Importantly, the feasible and infeasible solutions that we get represent, respectively, *the lower and upper bounds on the optimal solution*. To obtain increasingly tighter bounds, we leverage the subgradient method – a robust technique that provides a policy for progressively updating the Lagrangian multipliers.

The above solution process is repeated until one of the three stopping criteria is met. The first criterion terminates the process when the LR and the obtained solutions differ by less than a given threshold, $\Delta$. Subsequently, the iterative process is stopped if the step size, determining the size of updates to the Lagrangian multipliers through the subgradient method, drops below a designated threshold $\Gamma$. Indeed, the step size is initially set to large values to facilitate rapid updates and then halved when the iterative process fails to improve the solution for $N$ iterations, aiming to refine and stabilize the overall process. The third stopping criterion finally sets a predefined maximum number of overall iterations, $\Lambda$.

### B. Problem relaxation and decoupling

To apply the LR to the xDeSh problem, we note that constraints (3), (6), and (7) entangle the service configuration

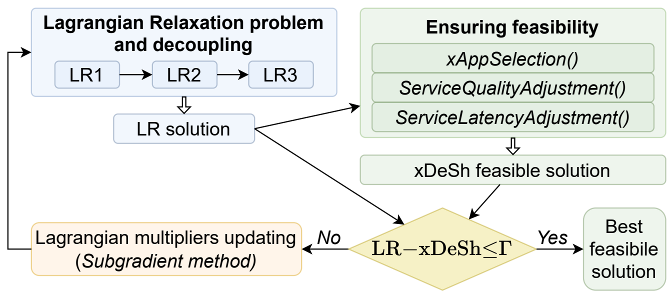
Lagrangian Relaxation problem and decoupling
LR1
LR2
LR3
LR solution
Ensuring feasibility
xAppSelection()
ServiceQualityAdjustment()
ServiceLatencyAdjustment()
xDeSh feasible solution
LR−xDeSh≤Γ
No
Yes
Lagrangian multipliers updating (Subgradient method)
Best feasibile solution

Fig. 4: The xDeSh problem is solved with an iterative algorithm that alternates the Lagrangian relaxation and the subgradient method until the set stopping criterion is met.

and the xApp selection subproblems. However, since the LR deals with inequalities, we split constraint (3) into:

$$\sum_{\chi\in\mathcal{X}_f}\sum_{j} v_{c_s,f_\chi^{(j)}} \geq z_{c_s}, \qquad \forall s\in\mathcal{S}, c_s\in\mathcal{C}_s, f\in\mathcal{V}_{c_s} \tag{10}$$

$$\sum_{\chi\in\mathcal{X}_f}\sum_{j} v_{c_s,f_\chi^{(j)}} \leq 1, \qquad \forall s\in\mathcal{S}, c_s\in\mathcal{C}_s, f\in\mathcal{V}_{c_s}\,. \tag{11}$$

The two inequalities above indeed provide, respectively, a lower and an upper bound on the number of xApps implementing the same function for a given service configuration $c_s$, and they collapse into (3) for the selected configuration. Moreover, we linearize (7), which links the configuration selection with the relative expected response latency by adopting the big-M linearization for each service $s$ implemented according to configuration $c_s$:

$$\tau_{c_s}(\{\boldsymbol{\rho}_{f_\chi^{(j)}} | \sum_j v_{c_s,f_{\chi f}^{(j)}}=1\}_{f\in c_s}) - T_s \leq M(1-z_{c_s})\,. \tag{12}$$

We relax constraints (10), (6) and (12) by introducing, respectively, the non-negative Lagrangian penalty terms $\boldsymbol{\beta}=\{\beta_{c_s,f}\}_{s,c_s,f}$, $\boldsymbol{\gamma}=\{\gamma_{c_s}\}_{s,c_s}$, and $\boldsymbol{\delta}=\{\delta_{c_s}\}_{s,c_s}$, which leads to the below LR formulation.

**xDeSh Lagrangian Relaxation (LR) Problem**

$$\max_{\boldsymbol{z},\boldsymbol{v},\boldsymbol{\rho}} \; \Psi_L(\boldsymbol{z},\boldsymbol{v},\boldsymbol{\rho},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\delta})$$

$$\text{s.t.} \; (2),(4),(5),(8),(9),(11)$$

$$z_{c_s} \in \{0,1\} \;\; \forall s\in\mathcal{S}, c_s\in\mathcal{C}_s$$

$$v_{c_s,f_{\chi_f}^{(j)}} \in \{0,1\} \;\; \forall c_s\in\mathcal{C}_s, f\in\mathcal{F}, \chi_f\in\mathcal{X}_f, j$$

$$\rho_{f_\chi^{(j)},k} \in [0,B_k] \;\; \forall k\in\mathcal{K}, f\in\mathcal{F}, \chi_f\in\mathcal{X}_f, j$$

In the above expression, the Lagrangian function $\Psi_L$ is defined as:

$$\begin{cases}
\Psi_L(\boldsymbol{z},\boldsymbol{v},\boldsymbol{\rho},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\delta}) = \Psi_{L,1}+\Psi_{L,2}+\Psi_{L,3} \\
\Psi_{L,1}(\boldsymbol{z},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\delta}) = \sum_{c_s} z_{c_s}(p_s-\gamma_{c_s}Q_s-M\delta_{c_s}- \\
\qquad\qquad \sum_{f\in c_s}\beta_{c_s,f}) \\
\Psi_{L,2}(\boldsymbol{v},\boldsymbol{\rho},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\delta}) = \sum_{c_s,f_\chi^{(j)}\in c_s}\beta_{c_s,f}v_{c_s,f_\chi^{(j)}}+\sum_{c_s}\gamma_{c_s}q_{c_s}- \\
\qquad\qquad \frac{1}{K}\sum_{f_\chi^{(j)},k\in\mathcal{K}_q}\frac{\rho_{f_\chi^{(j)},k}}{B_k} \\
\Psi_{L,3}(\boldsymbol{\rho},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\delta}) = -\sum_{c_s}\delta_{c_s}\tau_{c_s}-\frac{1}{K}\sum_{f,\chi,j,k\in\mathcal{K}_\tau}\frac{\rho_{f_\chi^{(j)},k}}{B_k}
\end{cases}$$

where we recall that $\mathcal{K}_\tau$ and $\mathcal{K}_q$ denote, respectively, the set of computing and storage resources.

Conveniently, the LR problem defined above can be easily decomposed into three (rather than two, as in [1]) independent subproblems by applying primal decomposition.

To enable LR primal decomposition, we first apply (8) to $\mathcal{K}_q$ and $\mathcal{K}_\tau$:

$$\sum_{f\in\mathcal{F}}\sum_{\chi\in\mathcal{X}}\sum_{j}\rho_{f_\chi^{(j)},k} \leq B_k,\ \forall k{\in}\mathcal{K}_q\,, \tag{13a}$$

$$\sum_{f\in\mathcal{F}}\sum_{\chi\in\mathcal{X}}\sum_{j}\rho_{f_\chi^{(j)},k} \leq B_k,\ \forall k{\in}\mathcal{K}_\tau\,. \tag{13b}$$

Similarly, we also split (9) into:

$$\sum_{f_\chi^{(j)}\in\mathcal{F}_1\setminus\mathcal{F}_2}\hat{\rho}_{f_\chi^{(j)},k} + \sum_{f_\chi^{(j)}\in\mathcal{F}_2}\rho_{f_\chi^{(j)},k}{\leq}B_k,\ \forall k{\in}\mathcal{K}_q\,, \tag{14a}$$

$$\sum_{f_\chi^{(j)}\in\mathcal{F}_1\setminus\mathcal{F}_2}\hat{\rho}_{f_\chi^{(j)},k} + \sum_{f_\chi^{(j)}\in\mathcal{F}_2}\rho_{f_\chi^{(j)},k}{\leq}B_k,\ \forall k{\in}\mathcal{K}_\tau\,. \tag{14b}$$

Hence, we derive the following three independent LR subproblems.

**LR1 problem.** LR1 tackles the problem of service configuration selection, breaking it down into a set of “0-1 knapsack” problems, each specifically designed for an individual service within $\mathcal{S}$. The objective for each knapsack problem is to identify the service configuration in $\mathcal{C}_s$ that maximizes the expression $\Psi_{L,1}$.

**LR1 Problem**

$$\begin{aligned} \max_{\boldsymbol{z}}\ & \Psi_{L,1}(\boldsymbol{z},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\delta}) \\ \text{s.t.}\ & (2) \\ & z_{c_s}\in\{0,1\}\ \forall s\in\mathcal{S},c_s\in\mathcal{C}_s \end{aligned}$$

Despite the solution space size (namely, $|S||C_s|$), solving LR1 is straightforward due to the uniqueness of the selectable configuration.

**LR2 problem.** LR2 deals with xApp instantiation and the concurrent allocation of storage and memory space. This process is a variant of the single-layer incapacitated FLP, where xApps take the place of facilities and service configurations stand in for customers. In accordance with $\Psi_{L,2}$, the xApps are associated with instantiation and association costs measured by, respectively, $\boldsymbol{\rho}$ and $\boldsymbol{\beta}$. The offered quality $q_{c_s}$ represents the degree of satisfaction of service configuration $c_s$ with the associated xApps. The primary objective in LR2 is to determine the optimal set of xApps for instantiation, maximizing the trade-off between instantiation and association costs while considering the quality provided to the services $q_{c_s}$.

**LR2 Problem**

$$\begin{aligned} \max_{\boldsymbol{v},\boldsymbol{\rho}}\ & \Psi_{L,2}(\boldsymbol{v},\boldsymbol{\rho},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\delta}) \\ \text{s.t.}\ & (4),(5),(11),(13a),(14a) \\ & v_{c_s,f_{\chi_f}^{(j)}}\in\{0,1\}\ \forall c_s\in\mathcal{C}_s,f\in\mathcal{F},\chi_f\in\mathcal{X}_f,j \\ & \rho_{f_\chi^{(j)},k}\in[0,B_k]\ \forall k\in\mathcal{K}_q,f\in\mathcal{F},\chi_f\in\mathcal{X}_f,j \end{aligned}$$

The dimensionality of LR2 is determined by the product of the number of configurations, which are $|\mathcal{S}|$ by incorporating the pruning technique introduced in IV-C, and the number of functions $|\mathcal{F}|$. Although the evaluation of service quality $q_{c_s}$ introduces non-linearity, LR2 can be optimally solved in many practical cases within a reasonable time by linearizing the computation of offered quality and leveraging standard solvers, e.g., Gurobi and CPLEX. Alternatively, heuristic methods can be applied to solve LR2 in extensive scenarios. The approach proposed in [28] is notable for its swift execution. The heuristic algorithm systematically eliminates xApps from the solution, assessing potential savings through their exclusion. This process iterates until all constraints are met, or there are no more xApps whose deactivation improves the objective function. The devised heuristic approach, as demonstrated in our numerical evaluation, shows an average gap of approximately 3% compared to the optimum. Its complexity is in the order of $O(|\mathcal{F}||\mathcal{V}_{c_s}|)$, due to the need to evaluate the benefit of deactivating each xApp while keeping the others.

**LR3 problem.** Finally, LR3 focuses on the allocation of computing resources for the xApps execution.

**LR3 Problem**

$$\begin{aligned} \max_{\boldsymbol{\rho}}\ & \Psi_{L,3}(\boldsymbol{\rho},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\delta}) \\ \text{s.t.}\ & (13b),(14b) \\ & \rho_{f_\chi^{(j)},k}\in[0,B_k]\ \forall k\in\mathcal{K}_\tau,f\in\mathcal{F},\chi_f\in\mathcal{X}_f,j \end{aligned}$$

LR3, whose complexity scales with the number of xApps (i.e., it is $O(|\mathcal{F}|)$), is convex and efficiently solvable using standard solvers.

The overall LR solution, also referred to as the relaxed solution, is obtained by combining the solutions of LR1, LR2 and LR3 subproblems. We can then prove the following proposition:

**Proposition 2.** *The solution provided for LR1, LR2 and LR3 problems provides a solution to the xDeSh Lagrangian Relaxation problem with an approximation ratio of 3.*

*Proof.* Heuristics with known approximation ratio exist for the knapsack ($(1-\epsilon)$ if the items size is within $\epsilon$ of the knapsack capacity [29]) and the uncapacitated FLP (3 using Primal-Dual methods [30]). Consequently, considering that (i) the LR solution is obtained by combining the LR1, LR2 and LR3 solutions, (ii) each service can have only one active configuration at a time (i.e., $\epsilon = 1$ in the knapsack), and (iii) neglecting the demand for service quality, the thesis holds. ■

## C. Variable pruning of the problem space

To further reduce the complexity of the LR solution, we incorporate relaxation and decoupling methods along with a pruning technique that decreases the dimensionality of LR2 and LR3 problems. We do so by identifying variables and constraints that do not contribute to the optimization process but rather add unnecessary complexity. As a result, we reduce the time complexity of the solution, without compromising its quality. In more detail, we apply pruning as follows:

**Algorithm 1** Ensuring feasibility

```
Input: $\{\bar{z}, \bar{v}, \bar{\rho}\}$                                  ▷ Relaxed solution.
Output: $\{\hat{z}, \hat{v}, \hat{\rho}\}$                                 ▷ Feasible solution.
1: $\hat{z} \leftarrow \bar{z}$            ▷ Accept relaxed service configuration choice.
2: if Eq. (10) is not respected for any service $s \in S$ then
3:     $\hat{v}_{c_s, f_\chi^{(j)}} \leftarrow$ xAppSelection algorithm    ▷ Fix the relaxed xApp selection.
4: if Eq. (6) is not respected for any service $s \in S$ then
5:     $\hat{v}_{c_s, f_\chi^{(j)}} \leftarrow$ ServiceQualityAdjustment    ▷ Increase the service quality by adjusting functions complexity.
6: if Eq. (12) is not respected for any service $s \in S$ then
7:     $\hat{\rho}_{f_\chi^{(j)},\text{cpu}} \leftarrow$ ServiceLatencyAdjustment    ▷ Reduce the service response latency by adjusting xApp CPU allocation.
8: while Eq. (8) or (9) are not respected do
9:     $s^* \leftarrow$ the lowest-priority implemented service with the highest deployment cost
10:    $\hat{z}_{c_{s^*}} \leftarrow 0$                         ▷ Deactivate service $\tilde{s}$
11:    $\hat{v}_{c_{s^*},\cdot} \leftarrow 0$            ▷ Turn-off the xApps utilised by $s^*$, only
```

**Algorithm 2** xAppSelection

```
Input: $\{\bar{z}, \bar{v}, \bar{\rho}\}$                                  ▷ Relaxed solution.
Input: $\{c_s\}$                               ▷ The service configuration at issue.
Output: $\{\hat{z}, \hat{v}, \hat{\rho}\}$                  ▷ Solution that satisfies Eq. (10) .
1: for every not provided $f \in c_s$ do
2:     if $\exists$ an xApp providing $f$ with the semantic required by $c_s$ then
3:         $(\chi^*, j^*) \leftarrow$ Identify the xApp with minimum load.
4:     else if $\exists$ an xApp providing $f$ then
5:         $(\chi^*, j^*) \leftarrow$ Identify the xApp that can be shared at minimum cost.
6:     else
7:         $(\chi^*, j^*) \leftarrow$ a new xApp providing $f$.
8:     $\hat{v}_{c_s, f_{\chi^*}^{j^*})} \leftarrow 1$            ▷ Share/instantiate the xApp $(f, \chi^*, j^*)$
```

•*Service configuration space pruning:* the LR2 search space can be effectively narrowed down by disregarding service configurations that were rejected in the LR1 solution, along with the exclusion of the xApps that are associated with such dropped configurations only. Throughout the LR2 resolution process, we deliberately exclude the potential association of xApps with non-deployed configurations, in line with (3). By doing so, the LR2 subproblem handles at most $|\mathcal{S}|$ service configurations.

•*xApp space pruning:* upon identifying the xApps accepted for implementation, a reduction in the size of LR3 becomes also feasible. Specifically, we reserve computational resources to the xApps that are implemented in the LR2 solution.

## D. Ensuring feasibility

As mentioned before, the solution of the LR1, LR2 and LR3 problems provide a solution to the xDeSh LR problem, which however may be unfeasible. We thus propose an *Ensuring feasibility* multi-stage algorithm to derive a feasible solution at a later step. As depicted in Fig. 4, upon receiving the relaxed solution, this algorithm identifies and properly rectifies any violation of the relaxed constraints (6), (10), and (12). The performed steps are reported in Alg. 1 and detailed below.

**1) Ensuring a compliant xApp selection.** The first stage of the *Ensuring feasibility* algorithm assesses the compliance of the LR solution with constraint (10). This involves verifying for each chosen service configuration if the xApps selected by the relaxed solution can meet the configuration functional

**Algorithm 3** ServiceQualityAdjustment

```
Input: $\{\bar{z}, \bar{v}, \bar{\rho}\}$                                  ▷ Relaxed solution.
Input: $\{c_s\}$                               ▷ The service configuration at issue.
Output: $\{\hat{z}, \hat{v}, \hat{\rho}\}$                  ▷ Solution that satisfies Eq. 6
1: while Eq. 6 not respected for $s$ and improvements are available do
2:     for each complexity factor increase for the xApps within $c_s$ do
3:         Compute the expected quality improvement.
4:         Compute the expected resource cost increase.
5:     $(\tilde{f}, \tilde{\chi}) \leftarrow$ Identify the xApp complexity improvement that leads the greatest service quality boost at minimum cost.
6: if Eq. (6) has been met then
7:     for every change in $c_s$ configuration $(f, \chi) \rightarrow (\tilde{f}, \tilde{\chi})$ do
8:         $\tilde{j} \leftarrow 1.$              ▷ Identify an existing or a new xApp instance
9:         $\hat{v}_{c_s, \tilde{f}_{\tilde{\chi}}^{(\tilde{j})}} \leftarrow 1$          ▷ Share/instantiate the xApp $(f, \tilde{\chi}, \tilde{j})$
10: else
11:     Remove service $s$.
```

requirements (Alg. 1, Line 2). If this condition is not met, the *xAppSelection* algorithm (Alg. 2) selects and adds to the LR solution the missing xApps (Alg. 1, Line 3). The *xAppSelection* algorithm evaluates whether the xApps already included in the solution can be shared. Precisely, sharing is preferred when the xApp performs the function with the same semantic as the concerned service, thereby not impacting the xApp load (Alg. 2, Lines 2-3). The same occurs when the additional computing resources needed to handle the extra xApp load are less than the resources required for a new xApp instance (Alg. 2, Lines 4-5). When multiple xApps meet a sharing criteria, the one with the lowest load is reasonably selected. Otherwise, a new xApp is istantiated (Alg. 2, Lines 6-7). Since the missing xApps for a service are $O(|\mathcal{V}_{c_s}|)$ (Alg. 2, Line 2), the complexity of the stage ensuring a compliant xApp selection is $O(|\mathcal{S}||\mathcal{V}_{c_s}|)$.

**2) Meeting service quality requirements.** The second stage of the *Ensuring feasibility* algorithm is dedicated to achieving the service quality targets. If any service fails to meet this criterion (Eq. (6)), our strategy incorporates the service configuration selection provided by the relaxed solution while simultaneously improving the complexity (hence, the output quality) with which the interested functions are deployed (Alg. 1, Lines 4-5).

The *ServiceQualityAdjustment* method (Alg. 3) is designed to achieve the specified task. The latter iterates through each function within the considered configuration, assessing any potential change in the deployment complexity level, until the service quality requirement is satisfied (Alg. 3, Lines 1-2). The potential complexity increments are hence $O(|\mathcal{V}_{c_s}|)$. For every complexity increase, the algorithm evaluates the gain in the expected service quality and the associated cost in resources (Alg. 3, Lines 3-4). Subsequently, the algorithm selects the xApp that offers the maximum enhancement in service quality at the minimal cost (Alg. 3, Line 5). Since the maximum number of interations of this first stage of the *ServiceQualityAdjustment* is $O(|\mathcal{V}_{c_s}|)$, the involved computational effort scales with $O(|\mathcal{S}||\mathcal{V}_{c_s}|^2)$. Once defined the new service deployment settings, the possible sharing of the xApp is assessed using the procedure outlined in Alg. 2 (Alg. 3, Lines 6-9). If the service quality requirement cannot be met, the service is dropped (Alg. 3, Lines 10-11).

**Algorithm 4** ServiceLatencyAdjustment

```
Input: $\{\bar{\boldsymbol{z}}, \bar{\boldsymbol{v}}, \bar{\boldsymbol{\rho}}\}$          ▷ Relaxed solution.
Input: $\{c_s\}$          ▷ The service configuration at issue.
Output: $\{\hat{\boldsymbol{z}}, \hat{\boldsymbol{v}}, \hat{\boldsymbol{\rho}}\}$          ▷ Solution that satisfies Eq. (12)
1: while Eq. (12) not respected for $s$ and improvements are available do
2:     for each xApp within $c_s$ do
3:         Compute the expected service response latency reduction by increasing by $\delta$ the xApp CPU allocation.
4:     $(\tilde{f}, \tilde{\chi})$ ← Identify the xApp CPU allocation increase that leads the greatest service response latency reduction.
5: if Eq. (12) has been met then
6:     Update $\bar{\boldsymbol{\rho}}$.
7: else
8:     Remove service $s$.
```

Ultimately, after confirming adequate quality for all services, it is evaluated whether of not if it is feasible to reduce the complexity of any function, thereby decreasing resource demand without violating the quality constraints of any service. If, in this process, the quality constraints of all services continue to be satisfied, the complexity reduction is applied (pseudocode is omitted for brevity).

**3) Meeting service response latency targets.** A similar approach is undertaken for service response latency. Specifically, the method *ServiceLatencyAdjustment* (Alg. 4) increases the CPU allocation for each xApp contributing to a service that fails to meet its response latency target. CPU allocation is increased first for the xApps for which the smallest CPU increase brings the best latency improvement (Alg. 4, Lines 2-4). The service is dropped whenever its response latency requirement cannot be met (Alg. 4, Lines 7-8). The complexity of this step of the above heuristic is $O(|\mathcal{S}||\mathcal{V}_{c_s}|)$. This differs from what discussed for the service quality, as the maximum number of iterations in the algorithm is restricted by the considered CPU allocation increment i.e., it is a constant.

**4) Meeting the resource budget.** The previous steps identify the xApps needed for the deployment of the requested services and adjust the compute resource allocation accordingly. However, the constraints (8)–(9) have to be fulfilled as well, i.e., it is imperative to verify the feasibility of the current solution and drop service requests as needed. The services to be discarded (if any) are those with the lowest priority and the highest deployment cost (Alg. 1, Lines 9-11). Evaluating the deployment cost for every service has complexity $O(|\mathcal{S}||\mathcal{V}_{c_s}|)$. This iterative procedure is repeated until the available budget is met by all resource types (Alg. 1, Line 8).

**5) The complexity of *Ensuring feasibility*.** Finally, the feasibility algorithm incurs a computational complexity that scales with $O(|\mathcal{S}||\mathcal{V}_{c_s}|) + O(|\mathcal{S}||\mathcal{V}_{c_s}|^2) + O(|\mathcal{S}||\mathcal{V}_{c_s}|) + O(|\mathcal{S}||\mathcal{V}_{c_s}|) = O(|\mathcal{S}||\mathcal{V}_{c_s}|^2)$ as its stages are sequentially independently executed The highest-complexity stage of the *Ensuring feasibility* algorithm is the *ServiceQualityAdjustment* method, which therefore determines the overall complexity of the *Ensuring feasibility* algorithm.

## E. The subgradient method

To penalize the violations of the relaxed constraints, the values of the Lagrangian multipliers can be determined so that the extent of such violations is minimized. The subgradient method is a viable and computationally efficient approach to solving this [31]. Specifically, it is an iterative optimization algorithm that generalizes the gradient descent algorithm for non-differentiable functions. It consists in iteratively updating the Lagrange multipliers in the direction of the subgradients of the LR problem objective function with respect to the Lagrange multipliers. The procedure's computation burden is proportional to the cardinality of the Lagrangian multipliers, which in our case are $O(|\mathcal{S}||\mathcal{C}_s||\mathcal{V}_{c_s}|)$. Importantly, the subgradient method is effective with non-smooth and non-convex functions [31], as is the case of the xDeSh problem.

## F. Overall complexity of the heuristic algorithm

We now provide a summary of the complexity of the different stages composing the proposed heuristic and evaluate that of the overall OREO algorithmic framework. In our analysis, we disregard inherently bounded parameters such as the maximum number of implementable instances of an xApp, or the available complexity levels with which a RAN function can be implemented.

First, as detailed in IV-B, LR2 is the Lagrangian subproblem that requires the highest computational effort, and its complexity scales as

$O(|\mathcal{F}||\mathcal{V}_{c_s}|)$ when the heuristic approach of [28] is utilized. Second, the complexity of the ensuring feasibility algorithm is equivalent to that of its most complicated step, which is $O(|\mathcal{S}||\mathcal{V}_{c_s}|^2)$. This effort is required to ensure that all services are provided with sufficiently high quality. Finally, the complexity of the subgradient method, used for updating the Lagrange multipliers, is $O(|\mathcal{S}||\mathcal{C}_s||\mathcal{V}_{c_s}|)$. Reasonably assuming that the number of configurations available for a service $|\mathcal{C}_s|$ is less than the functions involved in a configuration, i.e., $|\mathcal{V}_{c_s}|$, then this method's complexity scales down to $O(|\mathcal{S}||\mathcal{V}_{c_s}|^2)$.

Putting all the above together, the complexity of the proposed heuristic for solving the xDeSh problem can be expressed as $O(|\mathcal{F}||\mathcal{V}_{c_s}|) + O(|\mathcal{S}||\mathcal{V}_{c_s}|^2) + O(|\mathcal{S}||\mathcal{V}_{c_s}|^2) = O(|\mathcal{F}||\mathcal{V}_{c_s}| + |\mathcal{S}||\mathcal{V}_{c_s}|^2)$.

## V. Numerical Evaluation

We first evaluate OREO numerically, employing a custom-built Python simulator to assess its efficacy at scale. (An experimental validation is presented in Sec. VI).

Our numerical tests involve simulating MNOs that generate requests for a specific set of $N_s$ services. These service requests are forwarded to the non-RT RIC, where OREO strategically selects the best configuration from a set of $|\mathcal{C}_s|$ potential ones. Each configuration entails a maximum of $|\mathcal{V}_{c_s}|$ RAN functions from the pool of $|\mathcal{F}|$ available functions. Each function is deployable at three distinct levels of complexity at most. We evaluate four scenarios with varying scales (as outlined in Table II) representing increasingly complex xDeSh problem instances to be solved. These scenarios serve as synthetic, yet representative, benchmarks for assessing the nominal capability and scalability of OREO. Unless otherwise stated, results represent average values across 200 simulation runs with error bars indicating the standard deviation.

**Benchmarks.** We compare OREO against three alternatives:

TABLE II: Numerical test scenarios

| Scenario | $N_s$ | $\lvert\mathcal{C}_s\rvert$ | $\lvert\mathcal{F}\rvert$ | $\lvert\mathcal{V}_{c_s}\rvert$ |
|---|---|---|---|---|
| Small (**S**) | 4 | 2 | 4 | 3 |
| Medium (**M**) | 8 | 3 | 8 | 4 |
| Large (**L**) | 12 | 4 | 12 | 5 |
| Extra Large (**XL**) | 16 | 5 | 16 | 6 |

- The "Optimal" policy, which utilizes Gurobi to optimally solve the xDeSh problem;
- OREO-HC [1], the preliminary framework we presented in our conference paper, which solves the xDeSh problem by leveraging on Lagrangian relaxation and less meticulous decoupling methods, without applying any pruning technique. In this section, we refer to this benchmark as OREO-HC to highlight the higher complexity (HC) characterizing such an initial version of the framework relatively to the one we propose in this paper;
- OrchestRAN [11], a state-of-the-art O-RAN orchestrator that is the closest in terms of goals and underlying principles to OREO. The core function of OrchestRAN, similar to OREO's, is to determine the optimal selection of xApps to deploy according to high-level objectives while meeting desired latency requirements.

OrchestRAN does not model services as compositions of interconnected xApps. Hence, to support these cases, we let OrchestRAN deploy each services as a single xApp.

**Overall performance.** Table III compares the performance of each approach across all scenarios introduced in Table II. It reports the mean objective value ($\bar{\psi}$), worst-case performance relative to the Optimal policy ($\alpha$), 90% confidence interval ($0.9\alpha$), and average performance ratio ($\bar{\alpha}$). Additionally, the execution time ($t$) for each solution is provided, along with its ratio to the Optimal solver's execution time ($S$). Importantly, in Large (L) and Extra Large (XL) scenarios (where optimization variables exceed $10^4$), the Optimal policy cannot find a solution within a reasonable time frame. This makes performance comparisons with heuristic methods infeasible (indicated as "–" in the table). The same limitation applies to OREO-HC [1] for XL scenarios.

OREO consistently delivers solutions within (at least) 0.66 of the optimum across all tested scenarios but, more importantly, the estimated performance ratio increases to 0.97 within a 90% confidence interval. This narrow confidence interval demonstrates OREO's ability to maintain performance close to its mean, which consistently exceeds 0.99 in the tested scenarios. Furthermore, OREO's performance remains comparable to its preliminary version [1]. While the worst-case approximation ratio experiences a 32% reduction, we observe a minimal deterioration of only 2% within a 90% confidence interval, all while reducing execution time between $9\times$ and $48\times$, which enables scaling up to even larger scenarios (e.g., XL). In contrast, OrchestRAN exhibits a significant performance gap compared to the optimum. In the simplest scenarios (S and M), OrchestRAN achieves an approximation ratio that is, respectively, 83% and 14% worse than OREO.

**Deployed services and xApps.** We now gain more insights on the performance achieved by each solution by examining the percentage of services and xApps they provide. In Fig. 5(top), both the Optimal solution and OREO-HC [1] successfully deploy all services across scenarios S to M. Notably, OREO, a more agile version of OREO-HC [1], barely sacrifices the number of deployed services. In contrast, OrchestRAN exhibits a significant drop in service implementation, providing up to 14% fewer services than the optimum in small and medium-sized scenarios.

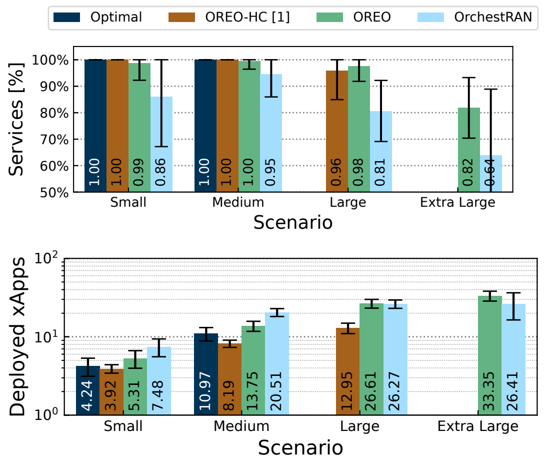

Fig. 5: Numerical results: Percentage of services (top) and xApps (bottom) deployed by Optimal, OREO, and OrchestRAN.

One contributing factor is evident in Fig. 5(bottom), which depicts the average number of xApps deployed. While OREO deploys more services than OrchestRAN, it does so with a lower number of xApp instances due to its superior xApp sharing and resource allocation abilities. Specifically, OREO instantiates an average of 17% fewer xApps (and up to 49.1%) than OrchestRAN. It deploys 20.2% to 39.3% more xApps than the Optimal solution, which leverages its xApps more effectively to offer a similar number of services. Importantly, however, the Optimal policy is very slow for medium scenarios and becomes impractical in Large scenarios (and larger) due to the problem's increased complexity. The same holds for OREO-HC in XL scenarios.

**Near-RT RIC resource consumption.** OREO's ability to share xApps across services significantly reduces resource consumption at the near-RT RIC compared to its benchmark, as detailed in Fig. 6. For instance, OREO saves 48.7% of CPU resources compared to OrchestRAN, while still offering a greater number of services. OREO, OREO-HC, and the Optimal policies exhibit comparable resource consumption, which scales with higher complexity scenarios. This highlights that the goal of minimizing resource usage is effectively balanced with other objectives within the xDeSh problem, such as maximizing the number of services deployed.

**Service requirements.** OREO effectively scales xApp resources based on the aggregated load of dependent services, ensuring that their requirements are successfully met. This is demonstrated in Fig. 7 and Fig. 8, which illustrate normalized service response latency (i.e., the ratio of actual to target service response latency) and normalized service quality across scenarios (i.e., the ratio of actual to target service quality). All solutions fulfill latency and quality targets, with average

TABLE III: Performance metrics of OREO and its benchmarks: i) mean objective value ($\bar{\psi}$); ii) worst-case performance ($\alpha$), 90% confidence interval ($0.9\alpha$), and average performance ratio ($\bar{\alpha}$) relative to the Optimal policy; iii) orchestrator's execution time ($t$) and its ratio to the Optimal solver's execution time ($S$)

| | **OREO** | | | | | | **OREO-HC [1]** | | | | | | **OrchestRAN** | | | | | | **Opt.** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\bar{\psi}$ | $\alpha$ | $0.9\alpha$ | $\bar{\alpha}$ | $t$ | $S$ | $\bar{\psi}$ | $\alpha$ | $0.9\alpha$ | $\bar{\alpha}$ | $t$ | $S$ | $\bar{\psi}$ | $\alpha$ | $0.9\alpha$ | $\bar{\alpha}$ | $t$ | $S$ | $t$ |
| **S** | 0.972 | 0.66 | 0.97 | 0.99 | 0.12 | 0.35 | 0.987 | 0.97 | 0.99 | 0.99 | 5.76 | 16.94 | 0.828 | 0.11 | 0.62 | 0.70 | 0.01 | 0.03 | 0.34 |
| **M** | 0.976 | 0.83 | 0.98 | 0.99 | 1.79 | 0.02 | 0.982 | 0.98 | 0.99 | 0.99 | 16.34 | 0.17 | 0.914 | 0.71 | 0.90 | 0.92 | 0.22 | 0.002 | 95.15 |
| **L** | 0.957 | – | – | – | 15.93 | – | 0.941 | – | – | – | 226.31 | – | 0.833 | – | – | – | 5.12 | – | – |
| **XL** | 0.837 | – | – | – | 43.81 | – | – | – | – | – | – | – | 0.674 | – | – | – | 146.12 | – | – |

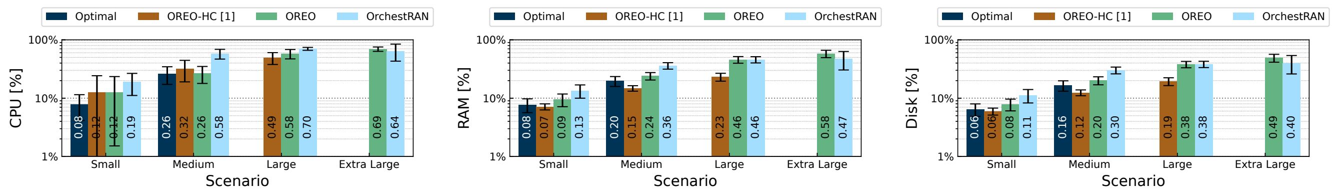

Fig. 6: Numerical results: CPU (left), RAM (center), and Disk (right) resources used by Optimal, OREO, and OrchestRAN.

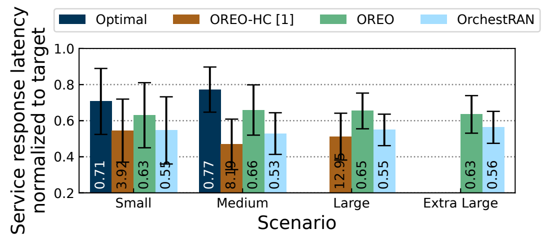

Fig. 7: Normalized response latency performance of RAN services offered by Optimal, OREO, and OrchestRAN.

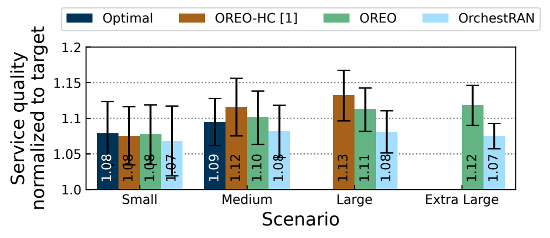

Fig. 8: Normalized service quality performance of RAN services offered by Optimal, OREO, and OrchestRAN.

latency-to-target ratios below 1 and average quality-to-target ratios above 1. Importantly, OREO achieves this with significantly lower resource consumption (as shown in Fig. 6). Further, OREO delivers superior service quality compared to OrchestRAN (2.1% improvement). This gain stems from OREO's ability to optimize xApp complexity levels to match the most demanding service among those sharing the xApp.

## VI. Experimental Validation

In this section, we first detail the experimental prototype of OREO on the Colosseum network emulator and outline the setup of the experiments (Sec. VI-A). Then we validate OREO through practical experiments that involve real-world RAN services and xApps (Sec. VI-B).

### A. Experimental setup

**Prototype on Colosseum.** We developed a proof-of-concept implementation of the OREO framework (see Fig. 1) using the Colosseum wireless network emulator [9]. Colosseum enables the emulation of diverse radio and computing scenarios, providing 128 Standard Radio Nodes (SRNs). These SRNs are x86-64 servers equipped with Ettus X310 radio frontends, interconnected through the Massive Channel Emulator (MCHEM). To create an O-RAN 5G network, we employed the SCOPE framework [32], an open-source platform specifically tailored for prototyping NextG systems based on srsRAN technology [33]. In our experimental setup, one SRN acts as the base station, utilizing the srsRAN CU/DU/RU for radio functions. This node also hosts the SMO framework and RIC components. Within the non-RT RIC, the OREO rApp (or its benchmark counterpart) receives service requests and optimizes xApp configurations and resource allocations for efficient service delivery. The SMO's xApp lifecycle manager then directs the instantiation of the selected xApps as Docker containers within the near-RT RIC. Although OREO is functionally standard-compliant, for the purpose of evaluation, we were unable to implement a fully standard-compliant deployment. Indeed, since some interfaces (e.g., O2 and HMI) are still in the process of standardization, we opted for a pragmatic approach that balances current capabilities with the evolving standards. Nonetheless, we are confident in the robustness and relevance of our results.

Our test RAN supports three predefined slices: eMBB for high-speed traffic (e.g., HD video), Machine-type Communication (MTC) for sensors and actuators, and URLLC for critical, low-latency applications. Each slice has a maximum capacity of 5 users, and user assignment to slices is static based on traffic type. User arrivals and departures in each slice follow a Poisson distribution with a rate of 0.05. We employ the *MGEN* tool[1] to model data transmission from the base station. Traffic flows are based on public datasets from a major Italian MNO, namely, Telecom Italia (TIM) in Milan, Italy [34] and they vary per slice: eMBB slices have a maximum load of 6.72 Mbps and a packet size of 1400 B; MTC slices have a maximum load of 0.512 Mbps and a packet size of 80 B; and

[1]https://github.com/USNavalResearchLaboratory/mgen

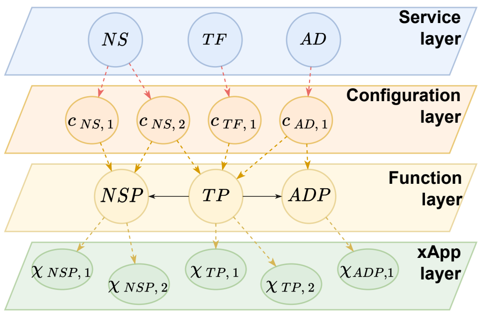

Fig. 9: Graph-based representation of the real-world RAN services integrated into the O-RAN platform hosted in the Colosseum testbed.

URLLC slices have a maximum load of 1.2 Mbps and a packet size of 200 B.

Finally, in our experiments we used a cell bandwidth of 5 MHz, which provides up to 25 Physical Resource Blocks (PRBs), and the 0-dB path loss static scenario in the Colosseum emulator. Each SRN is equipped with 2× 12-core Intel Xeon E5-2650 v4 processors and 128 GB of DDR4 memory. However, we limited xApp execution to 2 cores.

**RAN services and service configurations.** We designed three real-world RAN services (depicted in Fig. 9):

- Traffic Forecasting (TF): Predicts future user traffic loads for proactive RAN control. This service has a single configuration with a Traffic Predictor (TP) function.
- Network Slicing (NS): Dynamically allocates radio resources (PRBs) across slices. It offers two configurations: (i) NSP-only, which relies on near-RT RIC telemetry data, and (ii) NSP + TP, which incorporates a TP function to anticipate future traffic for more informed resource allocation.
- Anomaly Detection (AD): Monitors base station traffic to detect anomalies (security threats, failures, etc.). It has a single configuration with a Traffic Predictor (TP) followed by an Anomaly Detection Policy (ADP) function.

**O-RAN functions and xApps.** As depicted in Fig. 9, the aforementioned services can leverage three functions: the Traffic Predictor (TP), the Network Slicing Policy (NSP), and the Anomaly Detection Policy (ADP) functions, detailed in Table IV. TP consists of a Long Short-Term Memory (LSTM) model forecasting future traffic load based on past traffic samples. NSP uses an on-policy Reinforcement Learning (RL)-based slicing policy derived from [13] to dynamically allocate PRB to network slices. Based on traffic predictions, ADP detects network anomalies when the actual and expected behaviors of the traffic deviate significantly (see [35]). We trained each xApp either online, such as in the case of DRL-based NSP, or through datasets collected in Colosseum.

These functions can be combined to offer various RAN services. Table IV presents the xApps implementing these functions, along with their respective profiles. Importantly, when measuring disk utilization, we focus exclusively on the runtime occupation of the Docker container's writable layer. This approach excludes the disk space used by the Docker image itself, as the image is a prerequisite for building the Near-RT RIC's xApp catalog, regardless of whether a specific xApp is actively instantiated or not.

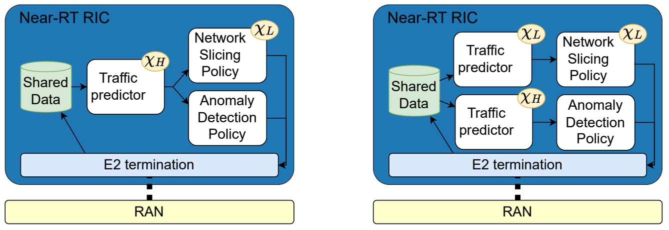

Fig. 10: Test scenario #1: service configuration under OREO (left) and OrchestRAN (right).

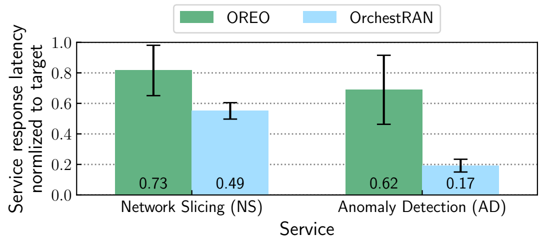

Fig. 11: Test scenario #1: normalized service response latency of OREO (green) and OrchestRAN (blue).

## B. Experimental results

We consider three distinct scenarios, detailed in Table V. Each scenario presents MNOs requesting varying service sets with diverse target quality and latency requirements. Since decision updates are needed every second, the arrival rate ($\lambda$) for each xApp is set to 1.

**Test scenario #1.** In this scenario, the MNO requests the NS service (quality target: 0.5, latency: 75 ms) and the AD service (quality target: 4.0, latency: 30 ms). Fig. 10 visually compares OREO's and OrchestRAN's configuration decisions. Both frameworks select the highest complexity TP function ($\chi_H$) for the AD service to meet its stringent quality target. However, OREO and OrchestRAN differ in NS configuration. OREO's holistic view of RAN services enables it to share the TP function, while OrchestRAN's approach necessitates two separate TP xApps.

This sharing strategy directly translates to resource savings, as shown in Table VI. Specifically, OREO exhibits a 66.7% reduction in computational resource allocation compared to OrchestRAN. This stems from OREO's ability to avoid over-provisioning and dynamically scale xApp resources based on service load and requirements. Additionally, OREO reduces RAM usage by 20.4% due to deploying fewer xApps. Disk space savings are also observed, though less significant.

Concerning service requirements, on the one hand, Fig. 11 illustrates the average value and the 95% confidence interval of the normalized (i.e., the ratio of actual to target) service response latency. Both OREO and its benchmark meet the service requirements. However, OrchestRAN struggles to balance meeting service requirements and resource consumption. Compared to OrchestRAN, OREO instead achieves a service latency on average 164% closer to its target, further highlighting its effectiveness.

On the other hand, Fig. 12 presents the average performance trend and the corresponding confidence intervals of the KPI for each of the 3 slices, respectively. We focus on downlink throughput, physical Transport Block (TB) count, and down-

TABLE IV: Description of the considered xApp functions: i) no. of available complexity levels and their design choice, ii) provided trade-offs, iii) storage (RAM, Disk) resource requirements, and iv) provided performance scores

| **Function** | **Complexity levels** | **Trade-offs (Accuracy vs. Data/Cycle)** | **Resource (RAM, Disk) requirements** | **Performance score** |
|---|---|---|---|---|
| Traffic Predictor (TP) | 2 options (24 - 30 Input samples; 2 - 3 LSTM layers; 79 - 90 Hidden layer size) | Higher complexity = Better accuracy, less data processed per CPU cycle | 180.6 - 181.1 MiB RAM, 4.46 kB Disk | Accuracy: 3.3 - 4.2 normalized MSE |
| Network Slicing Policy (NSP) | 2 options (Neural network layers) | - | 524.1 - 537.1 MiB RAM, 79.9 kB Disk | Reward: 0.0 - 0.18 (standalone), 0.59 - 0.80 (low-complexity TP), 0.67-1.00 (high-complexity TP) |
| Anomaly Detection Policy (ADP) | 1 option | Not applicable | 61.5 MiB RAM, 4.3 kB Disk | Score: same as paired TP |

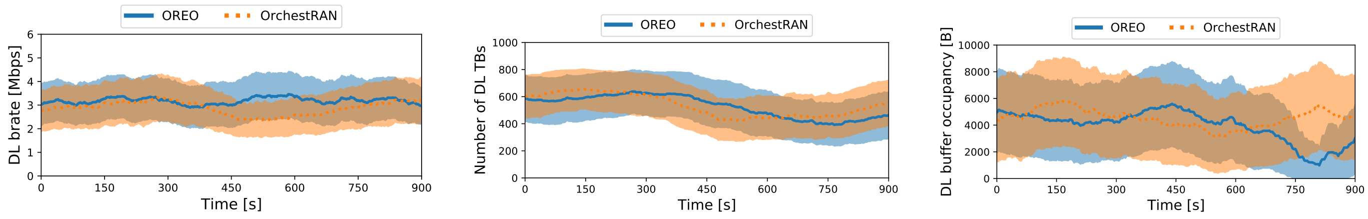

Fig. 12: Test scenario #1: KPI evolution for the eMBB (left), MTC (center) and URLLC (right) slices under OREO (blue, solid) and OrchestRAN (orange, dashed).

TABLE V: Experimental test scenarios and requested service targets

| | | ***Service*** | | |
|---|---|---|---|---|
| | | **TF** | **NS** | **AD** |
| ***Scenario*** | **#1** | – <br> – | $Q_s$ : 0.5 <br> $T_s$ : 75 ms | $Q_s$ : 4.0 <br> $T_s$ : 30 ms |
| | **#2** | – <br> – | $Q_s$ : 1.0 <br> $T_s$ : 100 ms | $Q_s$ : 4.0 <br> $T_s$ : 30 ms |
| | **#3** | $Q_s$ : 4.0 <br> $T_s$ : 50 ms | $Q_s$ : 1.0 <br> $T_s$ : 100 ms | – <br> – |

TABLE VI: Experimental results: resource utilization

| **Scenario** | **Orchestrator** | **CPU [%]** | **RAM [MiB]** | **Disk [kB]** |
|---|---|---|---|---|
| **#1** | OREO | 0.30 | 705.2 | 84.26 |
| | OrchestRAN | 0.90 | 885.8 | 88.72 |
| **#2** | OREO | 0.45 | 718.2 | 84.36 |
| | OrchestRAN | 1.00 | 899.3 | 88.82 |
| **#3** | OREO | 0.45 | 718.2 | 84.36 |
| | OrchestRAN | 0.65 | 718.2 | 84.36 |

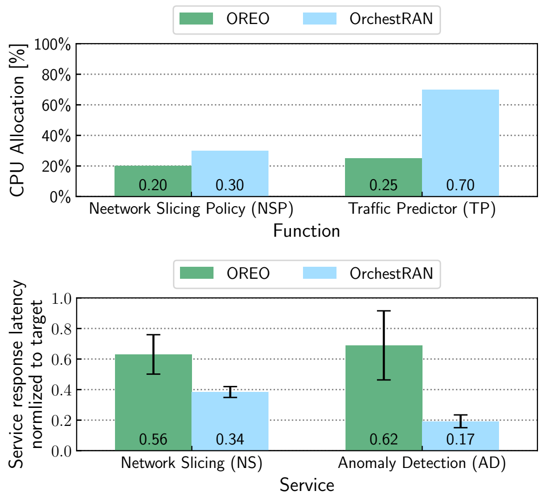

Fig. 14: Test scenario #2: xApp CPU allocation (top) and normalized service response latency (bottom) with OREO (green) and OrchestRAN (blue).

link buffer occupancy, following the methodology in [13]. OREO's NSP function significantly improves performance for eMBB and URLLC slices. eMBB users experience an 11.3% increase in average throughput, while URLLC users benefit from a 13.1% reduction in buffer occupancy, leading to lower queuing delays and latency. MTC slice performance remains largely unchanged. Importantly, these service quality gains are not due to changes in radio conditions, which are held constant for both solutions. Instead, they directly result from the NSP function's ability to leverage more accurate traffic forecasts provided by the TP function. Fig. 13 underlines this effect: OREO's NSP allocates more resources to the eMBB slice than OrchestRAN, especially during the traffic surge between 450 and 600 s. In summary, this result demonstrates that OREO's xApp sharing ability enables the use of higher-complexity function configurations, leading to superior service quality metrics without compromising resource efficiency.

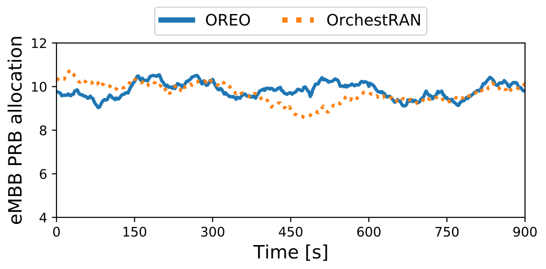

Fig. 13: Test scenario #1: physical resource blocks for the eMBB slice with OREO (blue, solid) and OrchestRAN (orange, dashed).

**Test scenario #2.** In the second scenario we tested, the MNO requests a higher quality score (1.0) and a stricter response latency target (100 ms) for the NS service, while the AD service requirements remain unchanged.

To meet these requirements, OrchestRAN deploys the NSP function at maximum complexity, differing from the configuration selected in the first scenario. In contrast, OREO maintains the service configuration shown in Fig. 10(left). Once again, OREO exhibits superior resource efficiency in RAN service deployment. As can be seen in Fig. 14 (top), it achieves a 55% reduction in CPU budget and allocates 20.1% less RAM compared to OrchestRAN (Table VI). These savings result from OREO's ability to share xApps between services and precisely scale xApp resources to meet latency targets, effectively avoiding over-provisioning (Fig. 14(bottom)).

**Test scenario #3.** In this case, the MNO requests the NS and TF services, thus replacing the AD service from previous scenarios. This change leads to both OREO and its competitor deploying the same set of xApps (Fig. 15(left)). The xApp chain consists of the TP and NSP functions, offering both TF and NS services and allowing for sharing.

Despite the identical xApp deployment, OREO shows a clear advantage in resource efficiency, especially in computational resources. This is highlighted in Table VI and Fig. 15(center), which illustrate the distribution of computational resources. Notably, OREO's ability to dynamically scale computational consumption based on target service latency (Fig. 15(right)) results in a significant 30% reduction in CPU usage.

## VII. Related Work

The O-RAN architecture facilitates the integration of network intelligence and automation within the RAN through a diverse range of third-party applications. Recently, significant focus has been placed on developing near-real-time RIC xApps [36], which serve various purposes such as analyzing, predicting, controlling, and automating behaviors within O-RAN networks. Examples of these xApps include those for network traffic classification [37], [38], network load forecasting [39], [40], setting policies to define RAN slices [13], [41], [42], and radio resources management [19], [43]. In this context, it becomes of paramount importance to effectively manage and orchestrate the RAN, as it is responsible for overseeing network intelligence management [44].

A significant body of research addresses the traditional challenges of RAN orchestration, offering strategies for optimizing energy consumption [45]–[47] and allocating RAN resources [48], [49]. In particular, [45] introduces a computationally efficient orchestrator designed for optimizing energy consumption in virtual RANs. Similarly, the ML-based approach outlined in [46] focuses on optimizing the allocation of both RAN radio and computing resources. Additionally, [48] and [49] tackle the orchestration and distribution of RAN resources to slices and users, respectively.

As for the management of RAN intelligence, this aspect has been scarcely addressed so far. It specifically demands optimal utilization of the diverse array of multi-vendor solutions [50] to meet the RAN service requirements set by MNOs. Solving the orchestration problem involves satisfying strict minimum requirements for service quality and latency, all while working within the constraints of limited resources available for deploying xApps and preventing conflicts between them. In this regard, [51] proposes a service platform designed to automate RAN orchestration and control. This platform supports the automated management and deployment of AI-driven closed-loop services, while also fulfilling service level agreements. In contrast, [52] introduces a distributed and dynamic policy for allocating and instantiating inference models. This approach aims to guarantee the completion of inference requests while trading off latency with accuracy. The study in [53] introduces an auto-scaling framework for O-RAN systems that utilizes data-driven latency models to allocate O-RAN applications across O-Cloud servers. Its main goal is to ensure that target latency requirements are met while maximizing the utilization and monetization of the shared infrastructure. However, [51], [52] do not specifically address the O-RAN architecture, and as such, they do not adhere to O-RAN specifications. On the other hand, [53] proposes a method for distributing O-RAN applications within the O-Cloud, but it overlooks an essential aspect of the network intelligence orchestration problem, which is the assessment of MNOs intents. Indeed, [53] is not concerned with determining the most appropriate applications to meet operator intents.

Of particular relevance to our study is OrchestRAN, an innovative O-RAN orchestrator outlined in [11] and further evaluated in [54]. The core function of OrchestRAN is to determine the optimal selection of xApps to deploy, and their respective locations. This decision-making process is geared towards satisfying the service demands put forth by MNOs while simultaneously meeting target performance requirements. Unlike our approach, OrchestRAN [11] assumes that RAN services are monolithic, i.e., exclusively provided by a single xApp. This setup restricts the potential for sharing low-level operations among services, consequently overlooking the significant issue of RAN resource consumption – a major factor contributing to MNOs' OPerational EXpenditures (OPEX) [8], [45].

Finally, an earlier version of our work has been presented in our conference paper [1]. With respect to [1], this paper includes a lower-complexity solution framework and experimental results performed through an O-RAN emulator on the efficiency of OREO in allocating radio resources and in fulfilling the target values of the services KPIs.

## VIII. Conclusions

We propose OREO, an O-RAN-compatible orchestrator of xApp-based RAN services. Unlike prior work, OREO recognizes that RAN services can be provided through sets of elementary functions implemented as xApps, which can be shared among different services to save resources. Leveraging a comprehensive, yet tractable, system model, we formulated the xDeSh problem and proved its NP-hardness. Then, in light of the problem complexity, we presented an efficient heuristic solution that configures requested services, maximizes xApps sharing, and dynamically allocates resources for

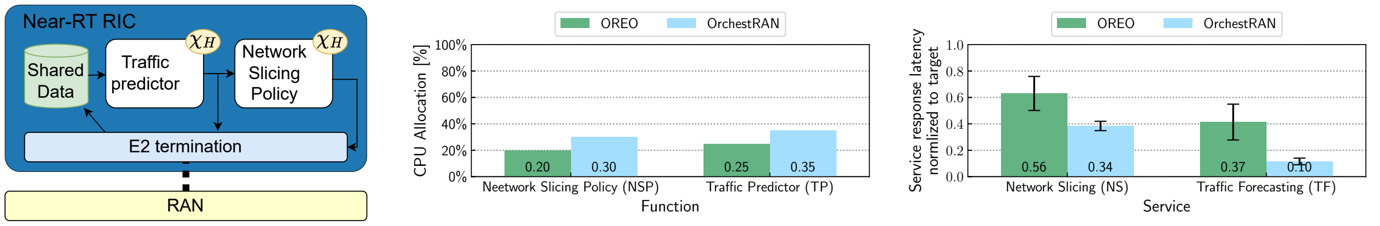

Fig. 15: Test scenario #3: service configuration (left), xApp CPU allocation (center) and normalized service response latency (right) under OREO (green) and OrchestRAN (blue).

xApp execution. To assess OREO performance, we conducted an extensive analysis and compared OREO's behavior to the optimum and state-of-the-art benchmarks. Our results demonstrate that OREO closely approximates the optimal solution, outperforming existing methods by allocating more services (on average, 16.2% and up to 35% in the largest scenario) while consuming less CPU (on average, 25.6% and over 31% in small-medium scenarios), all while meeting service requirements. We demonstrate that OREO provides robust scalability, enabling it to handle larger scenarios due to its low execution times. Furthermore, we developed an OREO prototype in the Colosseum network emulator, demonstrating its feasibility and seamless integration with the O-RAN architecture. Experimental results validate OREO's capabilities as it efficiently deploys xApps saving up to 66.7% of computing resources and improving the QoS with respect to the state of the art.

## References

[1] F. Mungari, C. Puligheddu, A. Garcia-Saavedra, and C. F. Chiasserini, "OREO: O-RAN IntElligence Orchestration for xApp-based Network Services," in *2024 IEEE Conference on Computer Communications (INFOCOM)*, 2024.

[2] Z. Zhang, Y. Xiao, Z. Ma, M. Xiao, Z. Ding, X. Lei, G. K. Karagiannidis, and P. Fan, "6G Wireless Networks: Vision, Requirements, Architecture, and Key Technologies," *IEEE Vehicular Technology Magazine*, vol. 14, no. 3, pp. 28–41, 2019.

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

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

[5] A. Garcia-Saavedra and X. Costa-Pérez, "O-RAN: Disrupting the Virtualized RAN Ecosystem," *IEEE Communications Standards Magazine*, vol. 5, no. 4, pp. 96–103, 2021.

[6] O-RAN Working Group 1, "O-RAN Use Cases Detailed Specification 14.0," O-RAN.WG1.Use-Cases-Detailed-Specification-R003-v14.00 Technical Specification, June 2024.

[7] ——, "O-RAN Use Cases Analysis Report 14.0," O-RAN.WG1.Use-Cases-Analysis-Report-R003-v14.00 Technical Specification, June 2024.

[8] K. Johansson, A. Furuskar, P. Karlsson, and J. Zander, "Relation between base station characteristics and cost structure in cellular systems," in *2004 IEEE 15th International Symposium on Personal, Indoor and Mobile Radio Communications (IEEE Cat. No.04TH8754)*, vol. 4, 2004, pp. 2627–2631 Vol.4.

[9] 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 *2021 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, 2021, pp. 105–113.

[10] M. Camelo, L. Cominardi, M. Gramaglia, M. Fiore, A. Garcia-Saavedra, L. Fuentes, D. De Vleeschauwer, P. Soto-Arenas, N. Slamnik-Krijestorac, J. Ballesteros, C.-Y. Chang, G. Baldoni, J. M. Marquez-Barja, P. Hellinckx, and S. Latré, "Requirements and Specifications for the Orchestration of Network Intelligence in 6G," in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, 2022, pp. 1–9.

[11] S. D'Oro, L. Bonati, M. Polese, and T. Melodia, "OrchestRAN: Network Automation through Orchestrated Intelligence in the Open RAN," in *2022 IEEE Conference on Computer Communications (INFOCOM)*, 2022, pp. 270–279.

[12] J. Zhang, Z. Wang, N. Ma, T. Huang, and Y. Liu, "Enabling efficient service function chaining by integrating NFV and SDN: Architecture, challenges and opportunities," *IEEE Network*, vol. 32, no. 6, pp. 152–159, 2018.

[13] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "ColO-RAN: Developing Machine Learning-based xApps for Open RAN Closed-loop Control on Programmable Experimental Platforms," *IEEE Transactions on Mobile Computing*, pp. 1–14, 2022.

[14] O-RAN Working Group 1, "O-RAN Architecture Description 12.00," June 2024.

[15] O-RAN Working Group 2, "O-RAN Non-RT RIC: Functional Architecture 1.01," O-RAN.WG2.Non-RT-RIC-ARCH-TR-v01.01 Technical Specification, June 2021.

[16] O-RAN Working Group 6, "Cloud Platform Reference Designs 2.0," February 2021.

[17] A. Lacava, M. Polese, R. Sivaraj, R. Soundrarajan, B. S. Bhati, T. Singh, T. Zugno, F. Cuomo, and T. Melodia, "Programmable and customized intelligence for traffic steering in 5g networks using open ran architectures," *IEEE Transactions on Mobile Computing*, vol. 23, no. 4, pp. 2882–2897, 2024.

[18] F. Gjeci, I. Filippini, and A. Capone, "An optimized handover management scheme tailored for heavy hitters in a disaggregated 5g o-ran architecture," in *2024 IFIP Networking Conference (IFIP Networking)*, 2024, pp. 647–653.

[19] S. Tripathi, C. Puligheddu, C. F. Chiasserini, and F. Mungari, "A Context-Aware Radio Resource Management in Heterogeneous Virtual RANs," *IEEE Transactions on Cognitive Communications and Networking*, vol. 8, no. 1, pp. 321–334, 2022.

[20] J. Garay, J. Matias, J. Unzilla, and E. Jacob, "Service description in the nfv revolution: Trends, challenges and a way forward," *IEEE Communications Magazine*, vol. 54, no. 3, pp. 68–74, 2016.

[21] I. Cohen, C. F. Chiasserini, P. Giaccone, and G. Scalosub, "Dynamic Service Provisioning in the Edge-cloud Continuum with Provable Guarantees," *arXiv preprint arXiv:2202.08903*, 2022.

[22] F. Ben Jemaa, G. Pujolle, and M. Pariente, "QoS-Aware VNF Placement Optimization in Edge-Central Carrier Cloud Architecture," in *2016 IEEE Global Communications Conference (GLOBECOM)*, 2016, pp. 1–7.

[23] R. Gouareb, V. Friderikos, and A.-H. Aghvami, "Virtual Network Functions Routing and Placement for Edge Cloud Latency Minimization," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2346–2357, 2018.

[24] J. Karkazis and T. Boffey, "The Multi-Commodity Facilities Location Problem," *Journal of the Operational Research Society*, vol. 32, no. 9, pp. 803–814, 1981.

[25] A. Klose, "A Lagrangean relax-and-cut approach for the two-stage capacitated facility location problem," *European journal of operational research*, vol. 126, no. 2, pp. 408–421, 2000.

[26] ——, "An LP-Based Heuristic for Two-Stage Capacitated Facility Location Problems," *Journal of the Operational Research Society*, vol. 50, no. 2, pp. 157–166, 1999.

[27] B. Gendron, P.-V. Khuong, and F. Semet, “A Lagrangian-Based Branch-and-Bound Algorithm for the Two-Level Uncapacitated Facility Location Problem with Single-Assignment Constraints,” *Transportation Science*, vol. 50, no. 4, pp. 1286–1299, 2016.

[28] J. M. M. Jesica de Armas, Angel A. Juan and J. P. Pedroso, “Solving the deterministic and stochastic uncapacitated facility location problem: from a heuristic to a simheuristic,” *Journal of the Operational Research Society*, vol. 68, no. 10, pp. 1161–1176, 2017.

[29] A. M. Frieze, M. R. Clarke *et al.*, “Approximation Algorithms for the m-Dimensional 0-1 Knapsack Problem: Worst-Case and Probabilistic Analyses,” *European Journal of Operational Research*, vol. 15, no. 1, pp. 100–109, 1984.

[30] K. Jain and V. V. Vazirani, “Approximation Algorithms for Metric Facility Location and k-Median Problems Using the Primal-Dual Schema and Lagrangian relaxation,” *Journal of the ACM (JACM)*, vol. 48, no. 2, pp. 274–296, 2001.

[31] M. Held, P. Wolfe, and H. P. Crowder, “Validation of Subgradient Optimization,” *Mathematical programming*, vol. 6, pp. 62–88, 1974.

[32] L. Bonati, S. D’Oro, S. Basagni, and T. Melodia, “Scope: an open and softwarized prototyping platform for nextg systems,” in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 415–426. [Online]. Available: https://doi.org/10.1145/3458864.3466863

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

[34] Telecom Italia, “Telecommunications - SMS, Call, Internet - MI,” 2015. [Online]. Available: https://doi.org/10.7910/DVN/EGZHFV

[35] J. Jiang and S. Papavassiliou, “Detecting network attacks in the internet via statistical network traffic normality prediction,” *Journal of Network and Systems Management*, vol. 12, pp. 51–72, 2004.

[36] O-RAN Working Group 3, “O-RAN Near-RT RAN Intelligent Controller Near-RT RIC Architecture 2.01,” March 2022.

[37] J. Shi, M. Liu, C. Hou, M. Jiang, and G. Xiong, “Online Encrypted Mobile Application Traffic Classification at the Early Stage: Challenges, Evaluation Criteria, Comparison Methods,” in *2021 IEEE 6th International Conference on Computer and Communication Systems (ICCCS)*, 2021, pp. 1128–1135.

[38] 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.

[39] A. M. Nagib, H. Abou-Zeid, H. S. Hassanein, A. Bin Sediq, and G. Boudreau, “Deep Learning-Based Forecasting of Cellular Network Utilization at Millisecond Resolutions,” in *ICC 2021 - IEEE International Conference on Communications*, 2021, pp. 1–6.

[40] 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.

[41] S. Bakri, P. A. Frangoudis, A. Ksentini, and M. Bouaziz, “Data-Driven RAN Slicing Mechanisms for 5G and Beyond,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4654–4668, 2021.

[42] J. Mei, X. Wang, K. Zheng, G. Boudreau, A. B. Sediq, and H. Abou-Zeid, “Intelligent Radio Access Network Slicing for Service Provisioning in 6G: A Hierarchical Deep Reinforcement Learning Approach,” *IEEE Transactions on Communications*, vol. 69, no. 9, pp. 6063–6078, 2021.

[43] 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*, vol. 39, no. 18, pp. 5706–5714, 2021.

[44] O-RAN Working Group 2, “O-RAN AI/ML workflow description and requirements 1.03,” October 2021.

[45] 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 *2021 IEEE Conference on Computer Communications (INFOCOM)*, 2021, pp. 1–10.

[46] J. A. Ayala-Romero, A. Garcia-Saavedra, M. Gramaglia, X. Costa-Pérez, A. Banchs, and J. J. Alcaraz, “vrAIn: Deep Learning Based Orchestration for Computing and Radio Resources in vRANs,” *IEEE Transactions on Mobile Computing*, vol. 21, no. 7, pp. 2652–2670, 2022.

[47] Z. Chang, Z. Han, and T. Ristaniemi, “Energy Efficient Optimization for Wireless Virtualized Small Cell Networks With Large-Scale Multiple Antenna,” *IEEE Transactions on Communications*, vol. 65, no. 4, pp. 1696–1707, 2017.

[48] S. Chatterjee, M. J. Abdel-Rahman, and A. B. MacKenzie, “On Optimal Orchestration of Virtualized Cellular Networks With Statistical Multiplexing,” *IEEE Transactions on Wireless Communications*, vol. 21, no. 1, pp. 310–325, 2022.

[49] S. Parsaeefard, R. Dawadi, M. Derakhshani, T. Le-Ngoc, and M. Baghani, “Dynamic Resource Allocation for Virtualized Wireless Networks in Massive-MIMO-Aided and Fronthaul-Limited C-RAN,” *IEEE Transactions on Vehicular Technology*, vol. 66, no. 10, pp. 9512–9520, 2017.

[50] A. Arnaz, J. Lipman, M. Abolhasan, and M. Hiltunen, “Towards integrating intelligence and programmability in open radio access networks: A comprehensive survey,” *Ieee Access*, 2022.

[51] X. Li, A. Garcia-Saavedra, X. Costa-Perez, C. J. Bernardos, C. Guimarães, K. Antevski, J. Mangues-Bafalluy, J. Baranda, E. Zeydan, D. Corujo, P. Iovanna, G. Landi, J. Alonso, P. Paixão, H. Martins, M. Lorenzo, J. Ordonez-Lucena, and D. R. López, “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, pp. 84–90, 2021.

[52] T. Si Salem, G. Castellano, G. Neglia, F. Pianese, and A. Araldo, “Towards Inference Delivery Networks: Distributing Machine Learning with Optimality Guarantees,” in *2021 19th Mediterranean Communication and Computer Networking Conference (MedComNet)*, 2021, pp. 1–8.

[53] S. Maxenti, S. D’Oro, L. Bonati, M. Polese, A. Capone, and T. Melodia, “ScalO-RAN: Energy-aware Network Intelligence Scaling in Open RAN,” *arXiv preprint arXiv:2312.05096*, 2023.

[54] S. D’Oro, L. Bonati, M. Polese, and T. Melodia, “OrchestRAN: Orchestrating Network Intelligence in the Open RAN,” *IEEE Transactions on Mobile Computing*, pp. 1–16, 2023.

**Federico Mungari** (Student Member, IEEE) received the PhD degree in Electrical, Electronics and Communication engineering from Politecnico di Torino in 2024. Since 2023, he has been collaborating as a researcher with CNIT, Italy. His research focuses on the design of next-generation mobile network technologies, Open RAN architectures, and the application of machine learning in telecommunications.

**Corrado Puligheddu** (Member, IEEE) is an Assistant Professor at Politecnico di Torino, Italy, since 2023. He received his Ph.D. in Electrical, Electronics and Communication Engineering from Politecnico di Torino in 2022.

**Andres Garcia-Saavedra** (Member, IEEE) received the PhD degree from UC3M, in 2013. He is a principal researcher with NEC Laboratories Europe. He joined Trinity College Dublin (TCD), Ireland, as a research fellow until 2015.

**Carla Fabiana Chiasserini** (Fellow, IEEE) is professor with Politecnico di Torino and EiC of Computer Communications. She was a visiting researcher with UCSD and a visiting professor with Monash University in 2012 and 2016.