uc3m | Universidad Carlos III de Madrid

e-Archivo

This is a postprint/accepted version of the following document:

Ayala Romero, J.A., García Saavedra, A., Gramaglia, M., Costa Pérez, X., Banchs, A. y Alcaraz, J.J. vrAIn: Deep Learning based Orchestration for Computing and Radio Resources in vRANs. *IEEE Transactions on Mobile Computing, 21(7), July 2022, Pp. 2652-2670*

DOI: https://doi.org/10.1109/TMC.2020.3043100

© 2020 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

vrAIn: Deep Learning based Orchestration for Computing and Radio Resources in vRANs

Jose A. Ayala-Romero, Andres Garcia-Saavedra, Marco Gramaglia,
Xavier Costa-Perez *Senior Member, IEEE,* Albert Banchs *Senior Member, IEEE,* and Juan J. Alcaraz

Abstract—The virtualization of radio access networks (vRAN) is the last milestone in the NFV revolution. However, the complex relationship between computing and radio dynamics make vRAN resource control particularly daunting. We present `vrAIn`, a resource orchestrator for vRANs based on deep reinforcement learning. First, we use an autoencoder to project high-dimensional context data (traffic and channel quality patterns) into a latent representation. Then, we use a deep deterministic policy gradient (DDPG) algorithm based on an actor-critic neural network structure and a classifier to map contexts into resource control decisions.
We have evaluated `vrAIn` experimentally, using an open-source LTE stack over different platforms, and via simulations over a production RAN. Our results show that: (i) `vrAIn` provides savings in computing capacity of up to 30% over CPU-agnostic methods; (ii) it improves the probability of meeting QoS targets by 25% over static policies; (iii) upon computing capacity under-provisioning, `vrAIn` improves throughput by 25% over state-of-the-art schemes; and (iv) it performs close to an optimal offline oracle. To our knowledge, this is the first work that thoroughly studies the computational behavior of vRANs and the first approach to a model-free solution that does not need to assume any particular platform or context.

Index Terms—RAN virtualization; resource management; machine learning

1 Introduction

Radio Access Network virtualization (vRAN) is well-recognized as a key technology to accommodate the ever-increasing demand for mobile services at an affordable cost for mobile operators [1]. vRAN centralizes *softwarized* radio access point (RAP)[1] stacks into computing infrastructure in a cloud location—typically at the edge, where CPU resources may be scarce. Fig. 1 illustrates a set of vRAPs sharing a common pool of CPUs to perform radio processing tasks such as signal modulation and encoding (red arrows). This provides several advantages, such as resource pooling (via centralization), simpler update roll-ups (via softwarization) and cheaper management and control (via commoditization), leading to savings of 10-15% in capital expenditure per km^2 and 22% in CPU usage [2], [3].

It is thus not surprising that vRAN has attracted the attention of academia *and* industry. OpenRAN[2], O-RAN[3] or Rakuten's vRAN—led by key operators (such as AT&T, Verizon or China Mobile), manufacturers (such as Intel, Cisco or NEC) and research leaders (such as Standford University)—are examples of publicly disseminated initiatives towards fully programmable, virtualized and open RAN solutions based on general-purpose processing platforms and decoupled base band units (BBUs) and remote radio units (RRUs).

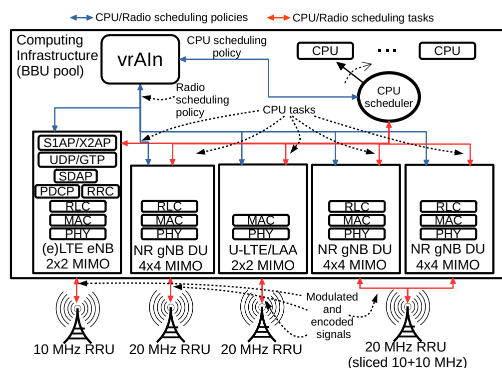

Fig. 1. `vrAIn`: A vRAN resource orchestrator

Despite the above, the gains attainable today by vRAN are far from optimal, and this hinders its deployment at scale. In particular, computing resources are inefficiently pooled since most implementations over-dimension computational capacity to cope with peak demands in real-time workloads [4], [5]. Conversely, *substantial cost savings can be expected by dynamically adapting the allocation of resources to the temporal variations of the demand across vRAPs* [3], [6]. There is nonetheless limited hands-on understanding of the computational behavior of vRAPs and the relationship between radio *and* computing resource dynamics. ***Such an understanding is required to design a practical vRAN resource management system—indeed the goal of this paper.***

Towards a cost-efficient resource pooling. Dynamic resource allocation in vRAN is an inherently hard problem:

(i) The computational behavior of vRAPs depends on many factors, including the radio channel conditions or users' load demand, that may not be controllable. More specifically, there is a strong dependency with the *context* (such as data bit-rate load and signal-to-noise-ratio (SNR) patterns), the RAP configuration (e.g., bandwidth, MIMO setting, etc.) and on the infrastructure pooling computing resources;

- *J. Ayala-Romero is with Trinity College Dublin.*
- *A. Garcia-Saavedra is with NEC Labs Europe.*
- *X. Costa-Pérez is with NEC Labs Europe, i2CAT Foundation and ICREA.*
- *M. Gramaglia and A. Banchs are with Universidad Carlos III de Madrid.*
- *J. J. Alcaraz is with Technical University of Cartagena.*

1. The literature uses different names to refer to different radio stacks, such as base station (BS), eNodeB (eNB), gNodeB (gNB), access point (AP), etc. We will use RAP consistently to generalize the concept.
2. https://telecominfraproject.com/openran/
3. https://www.o-ran.org/

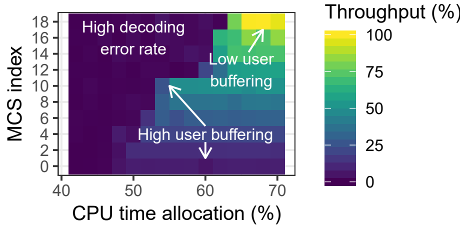

Fig. 2. A SISO 10-MHz LTE vRAP with maximum uplink traffic load and high SNR. High CPU and MCS allocations yield low data buffering (100% throughput). Low MCS allocation causes high user data buffering (<100% throughput). Low CPU time allocation renders high decoding error rate (≪100% throughput).

(ii) Upon shortage of computing capacity (e.g., with nodes temporarily overloaded due to orchestration decisions) CPU control decisions *and* radio control decisions (such as scheduling and modulation and coding scheme (MCS) selection) are coupled; certainly, it is well known that scheduling users with higher MCS incur in higher instantaneous computational load [5].

Let us introduce upfront some toy experiments to illustrate this. Note that we deliberately omit the details of our experimental platform (properly introduced in §4) to keep our motivation simple. We set up an off-the-shelf LTE user equipment (UE) and a vRAN system comprising `srsLTE`, an open-source LTE stack, over an i7-5600U CPU core @ 2.60GHz as BBU and a software-defined radio (SDR) USRP as RRU radio front-end. We let the UE transmit uplink UDP data at maximum nominal load with high SNR channel conditions and show in Fig. 2 the ratio of bits successfully decoded (throughput), where 100% denotes that all the demand is served, when selecting different MCS indexes (y-axis) and relative CPU timeshares (x-axis). The results yield an intuitive observation: higher modulation levels achieve higher performance, which in turn require larger allocations of computing resources. The reason is that vRAPs have tight deadlines to decode data, which are violated when there is not enough computing capacity available, producing decoding errors like low SNR does. This reduces throughput and, as a result, increases delay. ***This dependency motivates us to (i) devise novel algorithms to adjust the allocation of computing resources to the needs of a vRAN; and (ii) upon shortage of computing resources, explore strategies that make compute/radio control decisions jointly.***

Model-free learning. The aforementioned issues have been identified in some related research [5], [7], [8] (a proper literature review is presented in §8). Nevertheless, these works rely on models that need pre-calibration for specific scenarios and they do not consider the effect that different bit-rate patterns and load regimes have on computing resource utilization. *In reality, however, the relationship that system performance has with compute and radio policies is far from trivial and highly depends on the context (data arrival patterns, SNR patterns)* and *on the software implementation and hardware platform hosting the pool of BBUs.*

To emphasize the above point, we repeat the previous experiment for different SNR regimes (high, medium and low) and different mean bit-rate load regimes (10%, 30%, 50% and 70% of the maximum nominal capacity) for two different compute cores, the i7-5600U CPU core @ 2.60GHz used before and an i7-8650U CPU core @ 1.90GHz, and show in Fig. 3 (maximum load, variable SNR) and Fig. 4 (high SNR, variable load) the *relative throughput* with respect

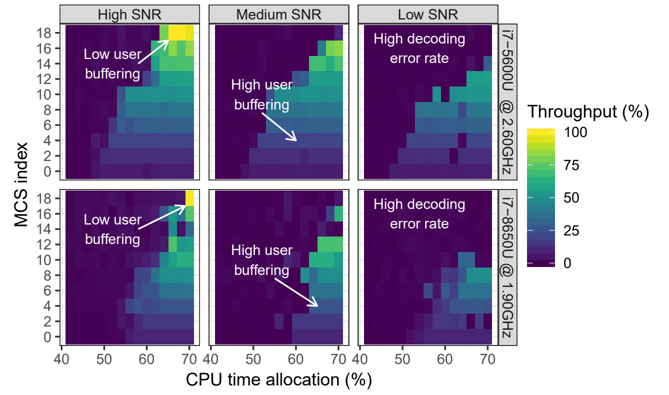

Fig. 3. vRAP with maximum uplink traffic load. Different computing platforms and SNR conditions yield different performance models.

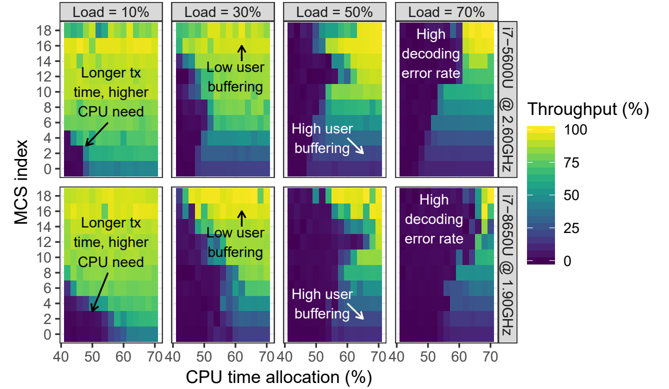

Fig. 4. vRAP with high SNR. Behavior is complex and non-linear. Light/dark areas (good/bad performance) follow irregular patterns.

to the load demand. The results make it evident that the system behavior shown in Fig. 2 substantially varies with the context (SNR, load) and the platform pooling computing resources, ***More importantly, the underlying model capturing this behavior is highly non-linear and far from trivial.***

All the above render tractable models in the literature (e.g., [5], [7], [8]) inefficient for practical resource control. Indeed, mechanisms based on such models are not able to accurately capture the complex behavior evidenced by our early experiments and hence perform poorly. We demonstrate this empirically in §6. *In contrast, we resort to model-free reinforcement learning methods that adapt to the actual contexts and platforms.* We present `vrAIn`, an artificial intelligence-powered (AI) vRAN controller that governs the allocation of computing and radio resources (blue arrows in Fig. 1).

The main *novel* contributions of this paper are as follows:

- We design a *deep autoencoder* that captures context information about vRAP load, signal quality and UE diversity time dynamics in a low-dimensional representation;
- We cast our control problem as a *contextual bandit* problem and solve it with a novel approach: (i) we decouple radio and computing control decisions to efficiently manage the high-dimensional action space; and (ii) we design a deep deterministic policy gradient (DDPG) algorithm for our contextual bandit setting to handle the real-valued nature of the control actions in our system;
- We implement a *proof-of-concept* of `vrAIn` using SDR boards and commodity computing nodes hosting software-based LTE eNB stacks, and assess its performance with a variety of scenarios and benchmarks.

To the best of our knowledge, our work is the first in the

literature that thoroughly explores empirically the computational behavior of a vRAN by means of an experimental setup. This paper extends our preliminary conference version [9] with the following contributions:

- We propose and evaluate an offline training method, suitable for real-life practical deployments;
- We simulate larger-scale scenarios to assess performance and cost savings in a real RAN deployment.

Our results *do not only shed light on the computational behavior of this technology across different contexts (radio and data traffic patterns), but also show that substantial gains can be achieved by developing autonomous learning algorithms that adapt to the actual platform and radio channel.*

2 Background

Prior to presenting the design of `vrAIn` (see §3), we introduce relevant information and notation used in the paper.

2.1 Radio Access Point

A Radio Access Point (RAP) implements the necessary processing stack to transfer data to/from UEs. These stacks may be heterogeneous, e.g., Fig. 1 shows 4G LTE, 5G NR, unlicensed LTE, and RAPs sharing a radio front-end (via *network slicing* [10], [11], [12]), and/or implement different functional splits [13], [14], [15], but they all share common fundamentals, such as OFDMA modulations and channel deecoders at the physical layer (PHY) that make `vrAIn` general across these vRAPs. Despite this heterogeneity, RAPs are typically dissected into 3 layers (L1, L2, L3).

L1 (PHY). We focus on sub-6GHz; specifically, on the uplink of 4G LTE and 5G NR since it is the more complex case as we have to rely on periodic feedback from users (while our implementation focuses on the uplink, our design applies to both uplink and downlink; the extension to downlink is straightforward as user buffers are local). L1 is implemented through a set of OFDMA-modulated channels, using a Resource Block (RB) filling across ten 1-ms subframes forming a frame. The channels used for data heavy-lifting are the Physical Uplink Shared Channel (PUSCH) and the Physical Downlink Shared Channel (PDSCH); usually modulated with QAM constellations of different orders (up to 256 in 5G) and MIMO settings, and encoded with a turbo decoder (4G) or LDCP code (5G). There are some differences between 4G and 5G PHYs, such as 5G's scalable numerology, but these are not relevant to `vrAIn`, which simply *learns* their computational behavior in a model-free manner. In brief, RBs assigned to UEs by the MAC layer are modulated and encoded with an MCS that depends on the user's Channel Quality Indicator (CQI), a measure of SNR that is locally available in the uplink and is reported periodically by UEs in the downlink. The scheme reported in [16] to map CQI values into MCSs is the most common approach and is blind to CPU availability.

L2 (MAC, RLC, PDCP). The MAC sublayer is responsible for (de)multiplexing data from/to different radio bearers to/from PHY transport blocks (TBs) and perform error correction through hybrid ARQ (HARQ). In the uplink, demultiplexing is carried out by the MAC scheduler by assigning RBs to UEs at every transmission time interval (TTI, usually equal to 1ms). Once this is decided, the RAP feeds the scheduling information to the UEs through a scheduling grant. 3GPP leaves the scheduler design open for vendor implementation. Moreover, the MAC layer also provides a common reference point towards different PHY carriers when using carrier aggregation. The higher sublayers (RLC, PDCP) carry out tasks such as data reordering, segmentation, error correction and cyphering; and provide a common reference point towards different PHY/MAC instances (e.g., from different vRAPs). Another L2 aspect relevant for the design of `vrAIn` are the Buffer State Reports (BSRs), which provide feedback to the RAPs about the amount of data each UE has pending to transmit. This information will be used by `vrAIn` to design a system state signal used for feedback on resource allocation decisions.

L3 (RRC, GTP). The Radio Resource Control (RRC) and GTP-U sublayers manage access information, QoS reporting and tunneling data between RAPs and the mobile core.

Notably, PHY (de)modulation/(de)coding operations consume most of the CPU cycles of the stack [17], which explains the dependency between CPU and MCS shown in §1. PDCP's (de)ciphering tasks consume most of the CPU cycles in L2 [18], albeit L2 is substantially less demanding than L1 [17]; furthermore, PDCP will be decoupled from the distributed unit (DU) in 5G (see NR gNB in Fig. 1).

2.2 Notation

We let $\mathbb{R}$ and $\mathbb{Z}$ denote the set of real and integer numbers, and $\mathbb{R}_+$ and $\mathbb{R}^n$ represent the sets of non-negative real numbers and n-dimensional real vectors, respectively. Vectors are in column form and written in bold font. Subscripts represent an element in a vector and superscripts elements in a sequence. For instance, $\langle \mathbf{x}^{(t)} \rangle$ is a sequence of vectors with $\mathbf{x}^{(t)} = (x_1^{(t)}, \dots, x_n^{(t)})^T$ being a vector from $\mathbb{R}^n$ and $x_i^{(t)}$ being the i'th component of the t'th vector in the sequence.

3 System Design

Fig. 5 illustrates a model of our system, where we can observe two functional blocks operating at different timescales:

- In the first block, **CPU schedulers** (which assign tasks to CPUs, e.g., subframes for decoding) and **radio schedulers** (which assign radio resources, e.g., selecting MCSs) operate at sub-millisecond timescales. `vrAIn` relies on simple computing and radio scheduling policies, which we introduce in §3.1, to influence their behavior.
- The second block is `vrAIn`, our vRAN resource orchestrator, a sequential decision-making entity that configures the above schedulers using, respectively, compute and radio scheduling policies over larger timescales. Our design is compliant with the architecture of O-RAN, which envisions a Non-Real-Time RAN Intelligent Controller operating at second-level granularity [19]. In practice, the operational timescale may be limited by the system constraints, such as the periodicity of feedback information from the users.

To overcome the issues mentioned in §1, our resource orchestrator consists of a feedback control loop where:

(i) *Contextual* information (SNR and data load patterns) is collected and encoded;
(ii) An *orchestrator that* maps contexts into computing and radio scheduling policies; and
(iii) A *reward* signal assesses the decisions taken and fine-tunes the orchestrator accordingly.

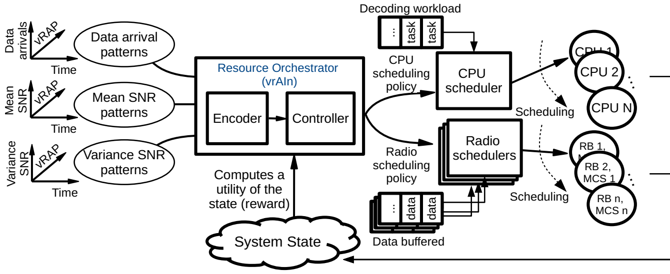

Fig. 5. System architecture.

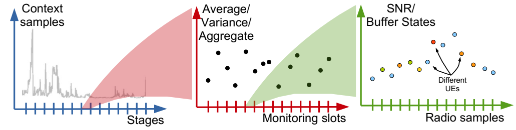

Fig. 6. Context samples, monitoring slots, and radio samples.

This falls naturally into the realm of **reinforcement learning** (RL) [20], an area of machine learning applied in human-level control (mastering games such as Go [21] or StarCraft II [22]), health-care [23] or finances [24]. Full-blown RL problems are usually modeled with Markov decision processes and use some model-free learning method (e.g., Q-learning) to estimate an action-value function [25]. However, *the impact that instantaneous actions (computing and radio scheduling policies) have on future contexts (radio signal quality patterns and user data demand), which RL usually captures with the recursive Bellman equation, is very limited in our case because of the different timescales between the schedulers and the resource orchestrator*. Thus, we can resort to a **contextual bandit** (CB) model, a type of RL applied in advertisement [26] or robot [27] control systems that can learn context-action mapping in a much simpler setup (without recursive action-value functions). There are several challenges to adopt a CB model, such as continuous and high-dimensional spaces, which we address in §3.2.

3.1 CPU and radio scheduling policies

CPU scheduling implies assigning tasks such as subframes to decode to an available CPU. In turn, radio scheduling involves deciding upon the number of RBs assigned to UEs, their location in frequency and time, their MCS and their transmission power. A plethora of computing and radio scheduling mechanisms [28], [29] have been proposed.

When orchestrating CPU and radio resources, our goal is both to provide *good performance*—minimizing data delivery delay—and make an *efficient resource usage*—minimizing CPU usage while avoiding decoding errors due to a deficit of computing capacity. To achieve these goals, when there is sufficient computing capacity, we can decode all frames with the maximum MCS allowed by the SNR conditions while provisioning sufficient CPU resources to this end. However, whenever there is a deficit of computing capacity, *we need to constraint the set of selected MCSs*, as otherwise we would incur into decoding errors that would harm the resulting efficiency. In this case, our approach is to limit the maximum eligible MCSs within each RAP when required, which has several advantages: (i) it is simple, as we only need to determine a single MCS bound for each RAP; and (ii) it provides fairness across UEs, reducing the performance of the UEs that are better off and preserving the less favorable ones. Thus, to orchestrate CPU and radio resources at vRAP i, `vrAIn` relies on the following scheduling policies:

- A *maximum fraction of time* $c_i \in \mathcal{C} := [0, 1] \subset \mathbb{R}$ allotted to a CPU (our **computing control decisions**); and
- A *maximum eligible MCS* $m_i \in \mathcal{M}$, where $\mathcal{M}$ is a discrete set of MCSs (our **radio control decisions**).

These control settings are configured by the resource orchestrator and can be easily implemented in any scheduler. Note that these are upper bounds, the CPU/radio schedulers still have the freedom to optimize the use of resources within these bounds. Our job is therefore to design a resource orchestrator that *learns* the behavior of *any* radio/CPU scheduler and maximizes performance using such interfaces.

3.2 vrAIn: vRAN resource orchestration

We hence formulate our resource control problem as a contextual bandit (CB) problem, a sequential decision-making problem where, at every time stage $n \in \mathbb{N}$, an agent observes a *context* or *feature* vector drawn from an arbitrary feature space $\mathbf{x}^{(n)} \in \mathcal{X}$, chooses an action $\mathbf{a}^{(n)} \in \mathcal{A}$ and receives a reward signal $r(\mathbf{x}^{(n)}, \mathbf{a}^{(n)})$ as feedback. The context $\mathbf{x}$ need not be stationary, as network conditions may change over time, and the sequence of context arrivals $\langle \mathbf{x}^{(n)} \rangle_{n\in\mathbb{N}}$ and the distribution E over context-reward pairs $(\mathbf{x}, r)$ are fixed and unknown *a priori*. Furthermore, we let $\pi(\mathbf{x}) : \mathcal{X} \to \mathcal{A}$ denote a deterministic orchestration function that maps contexts into actions i.e., scheduling policies, and

$$R_\pi := \mathbb{E}_{(\mathbf{x},\mathbf{r})\sim E}\Big[r\left(\mathbf{x}, \pi(\mathbf{x})\right) \Big] \tag{1}$$

denote the expected reward of a function π. The goal is to learn an optimal orchestration function $\pi^* := \arg\max_{\pi\in\Pi} R_\pi$ that maximizes instantaneous reward subject to $\sum_{i\in\mathcal{P}} c_i \leq 1$ to respect the system capacity, Π being the space of functions.

Context space. As shown by our early experiments in §1, SNR and traffic load are the contextual features that have most impact on the performance of a vRAP. We divide the time between two *stages* into $t := \{1, \ldots, T\}$ *monitoring slots*. At the end of each slot t, we aggregate *radio samples* collected during the last slot, as shown in Fig. 6.1, across all users in each vRAP $i \in \mathcal{P}$. Radio samples consist of: (i) the total amount of new bits pending to be transmitted $\delta_{i,n}^{(t)}$; (ii) mean SNR $\bar{\sigma}_{i,n}^{(t)}$; and (iii) variance SNR $\tilde{\sigma}_{i,n}^{(t)}$. This provides information about the *time dynamics* of the various variables of interest, namely (i) aggregate traffic load, (ii) the quality of the signals each vRAP has to process and (iii) the variability of the signal quality, which captures the impact of having multiple (heterogeneous) UEs in the vRAP in addition to their mobility. vRAPs can locally observe SNR data at high rate but the sampling rate of buffer state data is constrained to the periodicity of feedback from the users, which is also quantized. Conversely, for downlink traffic, vRAPs can locally observe buffer states, but only periodic and quantized data about channel quality. As a result, the interval between monitoring slots ($1/T$) depend upon such constraints. At the beginning of each stage n, we gather all samples into sequences of mean-variance SNR pairs and a sequence of load samples, and construct a *context sample* $x_i^{(n)} := \left\{ \langle\bar{\sigma}_{i,n}^{(t)}\rangle, \langle\tilde{\sigma}_{i,n}^{(t)}\rangle, \langle\delta_{i,n}^{(t)}\rangle \right\}_{t=\{1,\ldots,T\}}$ for vRAP i. Consequently, a *context vector* appends all context samples for all vRAPs, i.e., $\mathbf{x}^{(n)} = (x_i)_{\forall i\in\mathcal{P}} \in \mathcal{X} \subset \mathbb{R}^{3TP}$, where $P = |\mathcal{P}|$.

Action space. Our action space comprises all pairs of CPU and radio scheduling policies introduced in §3.1. Hence, $c_i^{(n)} \in \mathcal{C}$ and $m_i^{(n)} \in \mathcal{M}$ denote, respectively, the *maximum computing time share* (CPU scheduling policy) and the *maximum MCS* (radio scheduling policy) allowed to vRAP i in stage n. We also let $c_0^{(n)}$ denote the amount of CPU resource left unallocated (to save costs). Thus, a resource allocation action on vRAP i consists of a pair $a_i := \{c_i, m_i\}$ and a system action $\mathbf{a} = (a_i)_{\forall i \in \mathcal{P}} \in \mathcal{A} := \{(c_i \in \mathcal{C}, m_i \in \mathcal{M})\}_{\forall i \in \mathcal{P}}$.

Reward function. The objective in the design of `vrAIn` is twofold: (i) *when the CPU capacity is sufficient*, the goal is to minimize the operation cost (in terms of CPU usage) as long as vRAPs meet the desired performance; (ii) *when there is a deficit of computing capacity* to meet such performance target, the aim is to avoid decoding errors that lead to resource wastage, thereby maximizing throughput and minimizing delay. To meet this objective, we design the reward function as follows. Let q_{i,x_i,a_i} be the (random) variable capturing the aggregate buffer occupancy across all users of vRAP i given context x_i and action a_i at any given slot. As a quality-of-service (QoS) criterion, we set a target buffer size Q_i for each vRAP. Note that this criterion is closely related to the latency experienced by end-users (low buffer occupancy yields small latency) and throughput (a high throughput keeps buffer occupancy low). Thus, by setting Q_i, an operator can choose the desired QoS, which can be used to, e.g., provide differentiation across network slices. We let $J_i(x_i, a_i) := \mathbb{P}[q_{i,x_i,a_i} < Q_i]$ be the probability that q_{i,x_i,a_i} is below the target per vRAP i and define reward as:

$$r(\mathbf{x}, \mathbf{a}) := \sum_{i \in \mathcal{P}} J_i(x_i, a_i) - M\varepsilon_i - \lambda c_i \quad (2)$$

where ε_i is the *decoding error probability* of vRAP i (which can be measured locally), and M and λ are parameters that determine the weight of decoding errors and the trade-off between resource usage and performance, respectively. We set M to a large value to avoid decoding errors due to low allocations and λ to a small value to meet QoS requirements (while minimizing the use of compute resources).

Design challenges. `vrAIn`, our resource orchestrator illustrated in Figs. 5 and 7, is specifically designed to solve the above CB problem tackling the following two challenges:

(i) *The first challenge* is to manage the high number of dimensions of our contextual snapshots. We address this by implementing an **encoder** e that projects each context vector $\mathbf{x}$ into a latent representation $\mathbf{y} = e(\mathbf{x})$ retaining as much information as possible into a lower-dimensional space. The design of our encoder is introduced in §3.2.1.

(ii) *The second challenge* is the continuous action space. Recall that an action $\mathbf{a} \in \mathcal{A}$ comprises a (real-valued) compute scheduling policy $\mathbf{c} \in \mathcal{C}^P$ and a (discrete) radio scheduling policy $\mathbf{m} \in \mathcal{M}^P$. We design an **orchestrator** that decouples orchestration function $\pi(\mathbf{x}) : \mathcal{X} \to \mathcal{A}$ into two sequential orchestration functions:
- Radio orchestration function $\nu(\mathbf{y}, \mathbf{c}) = \mathbf{m}$, described in §3.2.3, which we design as a deep classifier that maps an (encoded) context $e(\mathbf{x})$ into a radio scheduling policy $\mathbf{m}$ that guarantees near-zero decoding error probability *given compute allocation* $\mathbf{c}$; and
- CPU orchestration function $\mu(\mathbf{y}) = \mathbf{c}$, described in §3.2.2, more challenging due to the continuous nature of $\mathcal{C}$, which we address with a deep deterministic policy gradient (DDPG) algorithm [30] that considers function ν as part of the environment to maximize reward.

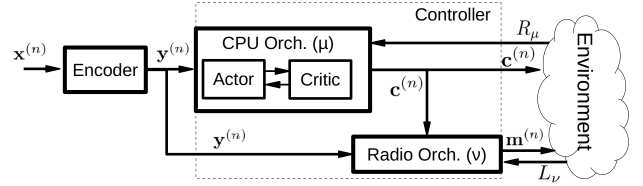

Fig. 7. Resource Orchestrator

While the above design decouples radio and compute orchestration functions, this does not affect the optimality of the solution. Indeed, as our radio orchestration function consists of a deterministic classifier that selects the most appropriate maximum MCS *for the allocation chosen by the CPU orchestration function*, when optimizing the CPU policy (allocation of compute resources), *we also optimize implicitly the radio policy* (maximum MCS).

In the following, we detail the design of `vrAIn`'s encoder (§3.2.1), radio orchestration function ν (§3.2.3) and CPU orchestration function μ (§3.2.2).

3.2.1 Encoder

Such a high-dimensional context space compounds our CB problem. Hence, we encode each context $\mathbf{x}^{(n)} \in \mathcal{X}$ into a lower-dimensional representation $\mathbf{y}^{(n)} \in \mathbb{R}^D$ with $D \ll \dim(\mathcal{X})$ with an *encoder* $e(\mathbf{x}^{(n)})$ (see Fig. 7).

Our context data consists of complex signals (in time and space) as they concern human behavior (communication and/or mobility patterns) and so, identifying low-dimensional handcrafted features is inherently hard. Moreover, useful representation functions may differ across scenarios. For instance, the average may be a good encoder of SNR sequences in low-mobility scenarios, a linear regression model such as Principal Component Analysis (PCA) [31] may be useful in high-mobility scenarios, and the variance may be needed in crowded areas. Therefore, there is no guarantee that hand-picked context representations are useful in general. Conversely, we resort to unsupervised representation learning. In particular, we focus on the Sparse Autoencoder (SAE), which is, to the best of our knowledge, the simplest known unsupervised learning approach that do not take the aforementioned assumptions, and is commonly used for such cases [32, Ch.14]. A SAE consists of two feed-forward neural networks: an encoder e_ξ (with an output layer of size D) and a decoder d_ψ (with an output layer of size $\dim(\mathcal{X})$) characterized by weights ξ and ψ, respectively. They are trained together so that the reconstructed output of the decoder is as *similar* as possible to the input of the encoder $\mathbf{x}$, i.e., $d(\mathbf{y}) = d(e(\mathbf{x})) \approx \mathbf{x}$.

A linear autoencoder, with linear activation functions in the hidden layers, will learn the principal variance directions (eigenvectors) of our contextual data (like PCA does). However, our goal is to discover more complex, multimodal structures than the one obtained with PCA, and so we use rectified linear units (ReLUs), and (ii) impose a sparsity constraint in the bottleneck layer (limiting the number of hidden units that can be activated) by adding a L1 regularization term to the loss function. Hence, we solve the following optimization problem during training:

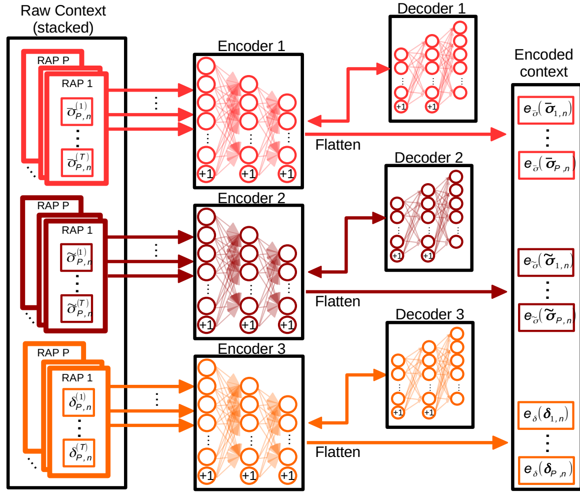

Fig. 8. Encoder design.

$$\arg\min_{\xi,\psi} \sum_{i=1}^{T} \frac{\|x_i - d(x_i)\|^2}{2T} + \omega \,\|\{\xi, \psi\}\| \tag{3}$$

where ω is a parameter of the L1 regularizer. This approach lets our encoder learn a code dictionary that minimizes reconstruction error *with minimal number of code words*.

Recall that $\mathbf{x}^{(n)} = \left(\langle\bar{\sigma}_{i,n}^{(t)}\rangle, \langle\tilde{\sigma}_{i,n}^{(t)}\rangle, \langle\delta_{i,n}^{(t)}\rangle\right)_{t=\{1,\ldots,T\},\forall i\in\mathcal{P}}$ consists of 3 different sequences. To avoid losing the temporal correlations within the sequences, we encode each of the three sequences independently, proceeding as follows:

(i) First, we train three different SAEs, one for each sequence comprising the triple $\left\{\langle\bar{\sigma}_{i,n}^{(t)}\rangle, \langle\tilde{\sigma}_{i,n}^{(t)}\rangle, \langle\delta_{i,n}^{(t)}\rangle\right\}$;

(ii) Second, we encode sequences corresponding to each individual vRAP i independently, i.e., $y_i = \{e_{\xi_k}(x_i)\}_{k=\{\bar{\sigma},\hat{\sigma},\delta\}}$;

(iii) Finally, we append all encoded sequences into a single vector $\mathbf{y} = (y_i)_{\forall i\in\mathcal{P}}$.

This approach, depicted in Fig. 8, avoids that the SAEs attempt to find correlations across vRAPs or sequences of different nature (SNR vs traffic load sequences) when optimizing the autoencoder parameters. As a result, `vrAIn` receives an encoded representation of the context $\mathbf{y}^{(n)} \in e(\mathcal{X})$ as input. To accommodate this in our formulation, we let $\hat{\pi} : \mathbb{R}^{(D_{\bar{\sigma}}+D_{\tilde{\sigma}}+D_{\delta})P} \rightarrow \mathcal{A}$ be the corresponding function mapping $\mathbf{y}^{(n)} = e(\mathbf{x}^{(n)})$ into an action in $\mathcal{A}$, with $D_{\bar{\sigma}}$, $D_{\tilde{\sigma}}$ and D_{δ} being the output layer of each of our encoders, and redefine $\hat{\Pi} = \{\hat{\pi} : \mathcal{X} \rightarrow \mathcal{A}, \pi(\mathbf{x}) = \hat{\pi}(e(\mathbf{x}))\}$.

3.2.2 Controller: CPU orchestrator (μ)

In the following, we design a function μ that determines the allocation of computing resources in order to maximize the reward function R defined in eq. (2). Note that R depends on both compute control decisions $\mathbf{c}$, and radio control decisions $\mathbf{m}$ (determined by function ν). We remark that, given $\mathbf{c}$, we can derive a deterministic orchestrator ν, which is presented later. As a result, when deriving the optimal CPU orchestration function we can simply treat ν as part of the environment. We hence redefine our reward function as:

$$R_{\mu} := \mathbb{E}_{(\mathbf{y},r)\sim E}\left[r\left(\mathbf{y}, \mu(\mathbf{y})\right)\right], \text{ with} \tag{4}$$

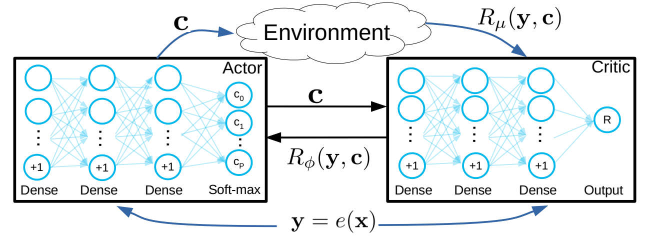

Fig. 9. CPU orchestration policy μ design.

$$r(\mathbf{y}, \mathbf{c}) = \sum_{i\in\mathcal{P}} J_i(y_i, c_i) - M\varepsilon_i - \lambda c_i \tag{5}$$

and $J_i(y_i, c_i) := \mathbb{P}\left[q_{i,y_i,a_i} < Q_i\right]$. Our goal is to learn an optimal function $\mu^* := \arg\max_{\mu} R_{\mu}$ subject to $\sum_{i=0}^{P} c_i = 1$ (note that c_0 denotes unallocated CPU time).

Since the above expectation depends only on the environment and a deterministic MCS selection function, we can learn R_{μ} off-policy, using transitions generated by a different stochastic exploration method. Q learning [25] is an example of a popular off-policy method. Indeed, the combination of Q learning and deep learning (namely DQNs [33]), which use deep neural network function approximators to learn an action-value function (usually represented by the recursive Bellman equation), has shown impressive results in decision-making problems with high-dimensional contextual spaces like is our case. However, DQNs are restricted to discrete and low-dimensional action spaces. Their extension to continuous domains like ours is not trivial, and obvious methods such as quantization of the action space result inefficient and suffer from the *curse of dimensionality*.

Instead, we resort to a deep deterministic policy gradient (DDPG) algorithm [30] using a model-free *actor-critic* approach, which is a reinforcement learning method successfully adopted in continuous control environments such as robotics [34] or autonomous navigation [35]. Our approach is illustrated in Fig. 9. We use a neural network μ_{θ} (the actor) parametrized with weights θ to approximate our deterministic compute orchestration function $\mu_{\theta}(\mathbf{y}) = \mathbf{c}$, and another neural network $R_{\phi}(\mathbf{y}, \mathbf{c})$ (the critic) parametrized with weights ϕ to approximate the action-value function R, which assesses the current function μ_{θ} and stabilizes the learning process. As depicted in the figure, the output of μ_{θ} (the actor) is a soft-max layer to ensure that $\sum_{i=0}^{P} c_i = 1$. Although they both run in parallel, they are optimized separately. The critic network needs to approximate the action-value function $R_{\phi}(\mathbf{y}, \mathbf{c}) \approx r\left(\mathbf{y}, \mu(\mathbf{y})\right)$ and to this end we use standard approaches such as the following update:

$$\Delta\phi = \beta\left(r\left(\mathbf{y}, \mu(\mathbf{y})\right) - R_{\phi}(\mathbf{y}, \mathbf{c})\right)\nabla_{\phi} R_{\phi}(\mathbf{y}, \mathbf{c}) \tag{6}$$

with learning rate $\beta > 0$. Regarding the actor, it is sufficient to implement an stochastic gradient ascent algorithm:

$$\nabla_{\theta} R_{\mu} \approx \mathbb{E}\left[\nabla_{\theta}\mu_{\theta}(\mathbf{y})\nabla_{\mathbf{c}} R_{\phi}(\mathbf{y}, \mathbf{c})\right] \tag{7}$$

Silver *et al.* [36] proved that this is the *policy gradient*. In this way, the actor updates its weights θ as follows:

$$\Delta\theta = \alpha\nabla_{\theta}\mu_{\theta}(\mathbf{y})\nabla_{\mathbf{c}} R_{\phi}(\mathbf{y}, \mathbf{c}) \tag{8}$$

with learning rate $\alpha > 0$.

In this way, we decouple scheduling policies and we can rely on the following radio orchestration function to maximize the reward defined in §3.2.

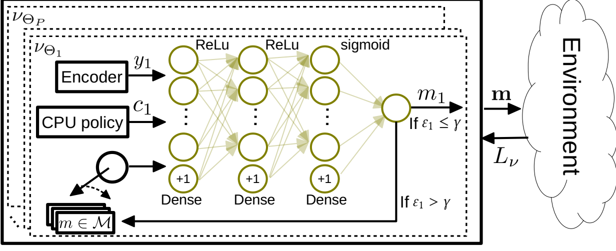

Fig. 10. Radio orchestrator ν design.

3.2.3 Controller: Radio orchestrator (ν)

In case the CPU capacity is insufficient to decode all the frames at the highest MCS allowed by the wireless conditions, we may need to impose radio constraints on some vRAP. To this end, our radio scheduling policy consists of imposing an upper bound $\mathbf{m}$ to the set of MCSs eligible by the radio schedulers such that the CPU load does not exceed capacity. Note that our radio scheduling policy will provide the highest possible $\mathbf{m}$ when there are no CPU constraints.

Following the above, we design a function ν that receives an encoded context $\mathbf{y}$ and a compute allocation $\mathbf{c}$ as input, and outputs a suitable radio scheduling decision $\mathbf{m}$. Our design consists of a simple neural network ν_{Θ_i} per vRAP i characterized by weights Θ_i with an input layer receiving (y_i, c_i, m_i), a single-neuron output layer activated by a sigmoid function and hidden layers activated by a ReLu function. We define γ as the threshold corresponding to the maximum acceptable decoding rate, which we set to a small value. Then, we proceed as follows to find the largest MCS satisfying this threshold. We train each ν_{Θ_i} as a classifier that indicates whether an upper bound MCS equal to m_i satisfies $\varepsilon_i \leq \gamma$ (in such a case m_i is an eligible bound for vRAP i as it ensures low decoding error rate given compute allocation c_i and context y_i) or $\varepsilon_i > \gamma$ (it is not). We use a standard loss function L_ν to train the classifiers with measurements of ε_i obtained at each stage n. In order to implement our function $\nu_{\mathbf{\Theta}} = \{\nu_{\Theta_i}\}_{i \in \mathcal{P}}$, we iterate, for each vRAP i, over the set of MCSs in descending order and break in the first m_i flagged by the classifier as appropriate ($\varepsilon_i \leq \gamma$), as shown in Fig. 10.

3.3 vrAIn system

`vrAIn`'s online operation is summarized in Algorithm 1. All neural networks are initialized (**steps (1)-(6)**) with random weights or pre-trained offline as shown in §**3.4**.

At the beginning of each stage n, `vrAIn`:

(i) Measures the empirical reward and decoding error rate of the previous stage, respectively, as $\tilde{r}^{(n-1)} := \sum_{i \in \mathcal{P}} \tilde{J}_i^{(n-1)} - M\tilde{\varepsilon}_i^{(n-1)} - \lambda c_i^{(n-1)}$ and $\tilde{\varepsilon}_i^{(n-1)}$(**step (8)**);
(ii) Stores $\{\mathbf{x}^{(n-1)}, \mathbf{y}^{(n-1)}, \mathbf{a}^{(n-1)}, \tilde{r}^{(n-1)}, \boldsymbol{\varepsilon}^{(n-1)}\}$ (**step (9)**);
(iii) Observes the current context $\mathbf{x}^{(n)}$ (**step (10)**).

Context $\mathbf{x}^{(n)}$ is first encoded into $\mathbf{y}^{(n)}$ in **step (13)**. Then, we use the actor network μ_θ to obtain $\mathbf{c}^{(n)}$ in **step (16)** and function ν to obtain $\mathbf{m}^{(n)}$ in **step (19)**. At last, `vrAIn` constructs action $\mathbf{a}^{(n)}$ for the current stage n in **step (20)**.

The encoders ($\{e_{\xi_k}, d_{\psi_k}\}_{k=\{\bar{\sigma},\hat{\sigma},\delta\}}$) and the radio classifiers ($\{\nu_{\Theta_i}\}_{\forall i \in \mathcal{P}}$) are trained every N_1 and N_3 stages with the last B_1 and B_3 samples, respectively (**steps (12)** and **(18)**). Conversely, function μ's actor-critic networks (μ_θ, R_ϕ) are trained every n with the last B_2 samples (**steps (14)-(15)**). Last, we implement a standard *online exploration method* that adds random noise $\eta^{(n)}$ to the actor's output with probability $\epsilon^{(n)}$, $\text{Bern}(\epsilon)$ being a Bernoulli-distributed variable with parameter ϵ. This online training method can be deactivated when `vrAIn` is in production.

Algorithm 1: `vrAIn` algorithm

1. Initialize autoencoders $\{e_{\xi_k}, d_{\psi_k}\}_{k=\{\bar{\sigma},\hat{\sigma},\delta\}}$ } Encoder
2. Set batch size B_1 and training period N_1
3. Initialize actor-critic networks μ_θ, R_ϕ } CPU orch.
4. Set batch size B_2 and exploration rate ϵ
5. Initialize classifiers $\nu_{\mathbf{\Theta}} = \{\nu_{\Theta_i}\}_{\forall i \in \mathcal{P}}$ } Radio orch.
6. Set batch size B_3 and training period N_3
7. **for** $n = \langle 1, 2, \ldots \rangle$ **do** *#Main Loop*
8. Measure reward $\tilde{r}^{(n-1)}$ and $\{\tilde{\varepsilon}_i^{(n-1)}\}_{i \in \mathcal{P}}$
9. Store $\left\{\mathbf{x}^{(n-1)}, \mathbf{y}^{(n-1)}, \mathbf{a}^{(n-1)}, \tilde{r}^{(n-1)}, \boldsymbol{\varepsilon}^{(n-1)}\right\}$
10. Observe context $\mathbf{x}^{(n)}$
11. **if** $\text{mod}(n, N_1) == 0$ **then** } Encoder
12. Update SAES $\{e_{\xi_k}, d_{\psi_k}\}_{k=\{\bar{\sigma},\hat{\sigma},\delta\}}$ using eq. (3) with B_1 samples
13. $\mathbf{y}^{(n)} \leftarrow e(\mathbf{x}^{(n)})$
14. Update critic R_ϕ using eq. (6) with B_2 samples } CPU orch.
15. Update actor μ_θ using eq. (8) with B_2 samples
16. $\mathbf{c}^{(n)} \leftarrow \mu_\theta(\mathbf{y}^{(n)}) + \text{Bern}(\epsilon^{(n)}) \cdot \eta^{(n)}$
17. **if** $\text{mod}(n, N_3) == 0$ **then** } Radio orch.
18. Update classifiers $\{\nu_{\Theta_i}\}$ using $L_\nu(\{\tilde{\varepsilon}_i\})$ with B_3 samples
19. $\mathbf{m}^{(n)} \leftarrow \nu_{\mathbf{\Theta}}(\mathbf{y}^{(n)}, \mathbf{c}^{(n)})$
20. $\mathbf{a}^{(n)} \leftarrow (\mathbf{c}^{(n)}, \mathbf{m}^{(n)})$ *#enforce action*

3.4 Offline pre-training

Reinforcement learning has traditionally been mistrusted for its application in production mobile network systems. Understandably, mobile operators cannot afford to have in production a system exploring low-performance control actions just for the sake of learning.

To address this issue, we propose an offline pre-training method comprised of (i) a *digital twin* of each individual vRAP, i.e., a digital replica such as a simulator or a model approximating the key performance indicators of the vRAN system; (ii) a *replay buffer* collecting prior experiences with the real system, and (iii) a method that uses a *batch* of samples obtained from our digital twin and/or our replay buffer, in each training episode.

This approach highly expedites the training process because (i) we can explore any (approximations of) context-action-reward tuple in a timely fashion using *digital twins*, (ii) we parallelize exploration by using *batches*, and (iii) we enable learning from real experience without using the time for gaining such experience by using a *replay buffer*. After using this offline pre-training method, `vrAIn` can be used in production without the need to explore online.

Replay buffer. We take advantage that our DDPG-based approach is off-policy, which allows us to store observations gained from past experiences with the real system in a replay buffer $\mathcal{D}$ and learn from them offline. However, it is inherently hard to provide a sufficiently rich dataset of experiences because of the large space of context and actions in our system, which becomes even more challenging to obtain as the number of vRAPs grows. We hence use a digital twin (details next) in combination with observations from our replay buffer to compensate for the gaps in $\mathcal{D}$.

Digital Twin. A digital twin is essentially a digital replica of a system that can be used in a controlled and safe environment for testing or, like in this case, for learning. This allows `vrAIn` to explore any context-action-reward tuple in a safe environment and without the need to gain the real experience, which takes time. This implies deriving a model (or building a simulator) that approximates the system reward as a function of any context-action vector pair. A digital twin can be built using different strategies. For instance, one can use a combination of models such as that in [7], queuing models and others to capture the relationship between contexts, computing policies, radio policies, decoding error rates and buffers occupancies (information needed to compute the system reward). However, this function is very complex due to the coupling of multiple vRAPs sharing a common computing pool; and becomes even more complex as the number of vRAPs increases.

Instead, we follow a simpler approach that stems from the observation that our reward function (defined in eq. (2)) is, in fact, the aggregate of individual rewards across vRAPs. This allows us to approximate the behavior of individual vRAPs *as opposed to approximating the behavior of the system as a whole*. To this aim, we use a data-driven modeling approach based on data collected from the real system. Specifically, we define $\tau_i : (y, c, m) \rightarrow (\tilde{J}_i, \tilde{\varepsilon}_i)$ as the digital twin of vRAP i, i.e., a function that maps encoded contexts and computing and radio controls (y, c, m) into an estimation of $\tilde{J}_i$ and decoding error $\tilde{\varepsilon}_i$. To do this, we construct two feed-forward neural networks $\tau_{i,\omega_1} \rightarrow (\tilde{J}_i)$ and $\tau_{i,\omega_2} \rightarrow (\tilde{\varepsilon}_i)$, with weights ω_1 and ω_2, respectively, trained with a dataset collected from the real system such that $\tau_i := (\tau_{i,\omega_1}, \tau_{i,\omega_2})$. The implementation details can be found in §4.3.

Batch Training. Further, we use batch training across multiple *episodes*. The overall logic is shown in Algorithm 2, where η is random noise (with any arbitrary distribution) and Uniform($\mathcal{X}$) is an instance of a uniform distribution in space $\mathcal{X}$. To this aim, at each episode k, we sample B random $(\mathbf{y}, \mathbf{c}, \mathbf{r})$ tuples comprised of:

- B_1 random observations from our replay buffer $\mathcal{D}$ (**steps (5)**-**(6)**); and
- B_2 random samples obtained with our digital twins τ_i (**steps (7)**-**(12)**);

Specifically, to obtain each random sample from our digital twins, we first choose a random context vector (**step (8)**), add random noise to our target function (**step (9)**), obtain the corresponding radio policy[4] (**step (10)**) and finally compute an approximation of the reward in **step (12)**.

The choice of B_4 and B_5 depends on the *richness* of the replay buffer, e.g., $B_4 \gg B_5$ is suggested for a rich $\mathcal{D}$. Nevertheless, once we have the collection of $B = B_4 + B_5$ samples for episode k, we update the actor and the critic weights (steps 13-14) of the whole batch at once using eqs. (6) and (8), accordingly. In this way, this batching method highly expedites the training process, especially if we used specialized hardware for training machine learning models, such as GPUs.

4. We assume the radio orchestrator and the autoencoders are previously trained offline following a similar approach as the one presented here to pre-train the computing orchestrator.

Algorithm 2: Actor-Critic Offline Pre-Trainer

```
1  Set batch size B = B_4 + B_5
2  Initialize actor-critic networks μ_θ, R_φ
3  for k = ⟨1, 2, ...⟩ do
4      for b = {1, 2, ..., B_4} do                              } Sample B_4
5          (y_b, c_b, m_b, J_b, ε_i) ← Uniform(D)                } context-action
6          r̃_b = Σ J_b − M ε_b − λ c_b                          } pairs from replay buffer
7      for b = {1, 2, ..., B_5} do                              } Sample B_5
8          y ← Uniform(Y)                                       } context-action
9          c_b ← μ_θ(y_b) + η                                   } pairs from
10         m_b ← ν_Θ(y_b, c_b)                                  } digital twins
11         (J̃_b, ε̃_b) = τ(y_b, c_b, m_b)
12         r̃_b = Σ J̃_b − M ε̃_b − λ c_b
13     Update critic R_φ using eq. (6) with B samples
14     Update actor μ_θ using eq. (8) with B samples
```

4 Prototype Implementation

Our vRAN testbed comprises one SDR USRP[5] per RAP as RRU radio front-end attached via USB3.0 to (i) a 2-core i7-5600U @ 2.60GHz compute node or (ii) a 4-core i7-8650U @ 1.90GHz compute node,[6] where we deploy our vRAP instances. Although there may be different approaches to implement a vRAP stack, it is reasonable to focus on open-source projects such as `OpenBTS`[7] (3G) and `OpenAirInterface`[8] or `srsLTE` [37] (4G LTE) to ensure reproducibility and *deployability*.

We build our experimental prototype around `srsLTE`'s `srseNB`. Note, however, that `vrAIn` takes no assumption about the underlying system and learns the relationship between context, scheduling policies and performance autonomously by interacting with the actual system. Hence, the design of the vrAIn approach is independent of the underlying platform and could be applied to any mobile system. Similarly, we deploy a UE per RAP,[9] each using one USRP attached to an independent compute node where an `srsLTE UE` stack runs (UEs do not share resources). Finally, with no loss in generality, we configure the vRAPs with SISO and 10 MHz bandwidth.[10] Let us summarize the design keys of `srsLTE eNB` in the sequel. The interested reader can revise a more detailed description in [37].

Fig. 11 depicts the different modules and threads implementing an LTE stack in `srsLTE eNB`. Red arrows indicate data paths whereas dark arrows indicate interactions between threads or modules. Every 1-ms subframe is assigned to an idle PHY DSP *worker*, which executes a pipeline that consumes most of the CPU budget of the whole stack [37], including tasks such as OFDM demodulation, PDCCH search, PUSCH/PUCCH encoding, PDSCH decoding, uplink signal generation and transmission to the digital converter. Having multiple DSPs allows processing multiple subframes in parallel. Since our computing infrastructure consists of 2 and 4-core processors, we set up a total number of 3 DSPs, which is sufficient since the HARQ process imposes a latency deadline of 3 ms (3 pipeline stages). The

5. USRP B210 from National Instruments/Ettus Research.
6. Intel Turbo Boost and hyper-threading are deactivated.
7. http://openbts.org/
8. https://www.openairinterface.org/
9. We use a single UE transmitting aggregated load (from several users)—note that `vrAIn` is scheduler-agnostic.
10. Note that the data-driven nature of `vrAIn` makes it agnostic to the MIMO or bandwidth configuration of the vRAPs.

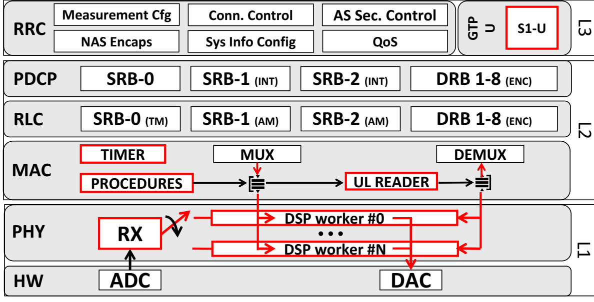

Fig. 11. Threading architecture in srsLTE. Boxes in red are threads.

remaining threads perform important operations that are less CPU demanding such as scheduling subframes to DSP workers (PHY RX) or procedures such as random access, uplink/downlink HARQ and scheduling data to physical resource blocks (MAC procedures), timer services (MAC timer), or pushing data from a buffer of uplink TBs to the upper layers (MAC UL reader).

In this way, a multi-thread process, which can be virtualized with virtual machines (like in [10]) or with Linux containers (LXCs), handles all the stack. `vrAIn` relies on the latter since it provides both resource isolation (through `namespaces`) and fine-grained control (through Linux control groups or `cgroups`) with minimal overhead. We next detail our platform's compute and radio control interfaces.

4.1 CPU scheduling policy

When allocating CPU resources to vRAPs, we follow a typical NFV-based approach [38] providing CPU reservations, which ensures isolation across different vRAPs.[11] We rely on Docker[12] for BBU isolation and fine-grained control of computing resources. Docker is an open-source solution that extends LXCs with a rich API to enforce computing resource allocations. Docker uses control groups (`cgroups`), a Linux kernel feature that limits, accounts for, and isolates resource usage of Linux processes within the group. Docker uses CFS (Completely Fair Scheduler) for CPU bandwidth control of `cgroups`. CFS provides *weight* based allocation of CPU bandwidth, enabling arbitrary slices of the aggregate resource. Hence, we implement a computing resource control action $c_i \in \mathcal{C}$ as a CFS CPU quota, which effectively upper bounds the relative CPU time allowed to each vRAP i. In detail, CFS allows the `cgroup` associated with the vRAP container to `cpu.cfs_quota_us` units of CPU time within the period of `cpu.cfs_period_us` (equal to 100 ms by default) by implementing a hybrid global CPU pool approach. More details can be found in [28].

In order for `vrAIn` to exploit Docker's CFS resource controller, we need to set the default scheduling policy of the DSP threads in `srsLTE eNB`, real-time by default, to `SCHED_NORMAL`, which is the default scheduling policy in a Linux kernel. This can be easily done with a minor modification to the PHY header files of `srsLTE eNB`. Moreover, it is worth remarking that, although our platform uses, for simplicity, Docker containers over a single compute node for resource pooling, `vrAIn` can be integrated into a multi-node cloud using, e.g, Kubernetes or Docker Swarm. In such cases, a compute control action $c_i \in \mathcal{C}$ requires Kubernetes or Docker Swarm to schedule vRAPs into compute nodes first, and then assign an appropriate CPU timeshare.

11. It is widely accepted in NFV that Virtual Network Functions (VNFs) need to have the required CPU resources reserved to ensure the proper operation of the network as well as to isolate VNFs that may belong to different actors (such as, e.g., different tenants in a network slicing context [12], [11]).

12. https://www.docker.com/

4.2 Radio scheduling policy

Unless otherwise stated, we focus on `srsLTE`'s uplink communication. Specifically, `srsLTE` allocates scheduling grants to UEs in a round-robin fashion and then computes their TB size (TBS) and MCS as follows. First, `srsLTE` maps the SNR into CQI using [16, Table 3]. Then, it maps the UE's CQI into *spectral efficiency* using 3GPP specification tables (TS 36.213, Table 7.2.3-1). Finally, it implements a simple loop across MCS indexes to find the MCS-TBS pair that better approximates such spectral efficiency. To this aim, `srsLTE` relies on an additional 3GPP specification table (TS 36.213, Table 7.1.7.1-1) to map an MCS index into a TBS.

A plethora of scheduling policies has been proposed (proportional fair, max-weight, etc. [29]). However, as explained in §3.1, `vrAIn` can learn the behavior of *any* low-level scheduler. We hence integrate our radio scheduling policy, we only need to write a handful of lines of code in `srsLTE`'s MAC procedures (see Fig. 11) to (*i*) upper bound the eligible set of MCSs with $m_i \in \mathcal{M}$—which we do by modifying the aforementioned MCS-TBS loop, and (*ii*) expose an interface to the orchestrator to modify $m_i \in \mathcal{M}$ online—which we do through a Linux socket.

4.3 Resource Orchestrator

We implemented `vrAIn` with the parameters shown in Table 1 using `Python` and `Keras`. In our implementation, the time between two stages takes 20 seconds and each context sample consists in $T = 200$ samples. We have tested other orchestration timescales[13] with marginal differences in performance. For this reason and due to space constraints, we focus our evaluation in other aspects of vrAIn.

CPU orchestrator. The CPU orchestrator μ consists of two neural networks (actor and critic), each with 5 hidden layers with $\{20, 40, 80, 40, 10\}$ neurons activated by a ReLu

13. A demonstration with $T = 100$, i.e., one stage every 10 seconds is available at https://youtu.be/1I8mcnHQcW8

TABLE 1
Parameters of our experimental prototype

	Hyperparameter	Value
Reward	λ	0.25
	M	2
	Q (bytes)	7000, 11000, 25000
Samples	T	200
	B_1, B_2, B_3	100
	N_1, N_3	∞ (trained offline)
Autoencoder	$D_{\tilde{\sigma}}, D_{\tilde{\sigma}}, D_{\tilde{\delta}}$	4
	ω	10^{-8}
	NN hidden layers	{ 100, 20 }
Radio orch.	γ	0.05
	NN hidden layers	$\{5, 8, 20, 30, 40, 40, 40, 40, 30, 20, 5\}$
CPU orch.	α, β	0.001
	$\epsilon^{(n)}$	0.995^n
	NN hidden layers	$\{20, 40, 80, 40, 10\}$

function. The actor has an input size equal to $\dim(\mathcal{Y})$, and output size equal to $P+1$ activated with a soft-max layer guaranteeing that $\sum c_i = 1$. In contrast, the critic has an input size equal to $\dim(\mathcal{Y})+P+1$ to accommodate the input context and the CPU scheduling policy, and an output size equal to 1 to approximate reward. Both neural networks are trained using Adam optimizer [39] with $\alpha = \beta = 0.001$ and a mean-squared error (MSE) loss function. Finally, unless otherwise stated, we set $M = 2$, $\lambda = 0.25$ and $\epsilon^{(n)} = 0.995^n$, which reduces exploration over time, when online.

Radio orchestrator. We implement our radio orchestrator ν with a set of P neural networks, each with 11 hidden layers of sizes $\{5, 8, 20, 30, 40, 40, 40, 40, 30, 20, 5\}$. We pre-train them using the dataset mentioned below with Adam optimizer and use a binary cross-entropy loss function L_ν, typical in classification problems.

Encoder. Our results in §6 indicate that 4 encoded dimensions represent a good trade-off between low dimensionality and reconstruction error. Hence, we implement each encoder network with 3 hidden layers of size $\{100, 20, 4\}$ (mirrored for the decoders); that is, each 200-sampled raw context is encoded into a 4-dimensional real-valued vector and appended together as shown in Fig. 8. We pre-train our encoders using Adam algorithm to minimize eq. (3) with $\omega = 10^{-8}$, and the dataset introduced next.

Complexity. The design of our context space, which aggregates samples across time and users within one stage, allows us to build a mechanism whose complexity is independent of the number of users and the dynamics within each stage. Complexity of `vrAIn` does grow with the number of vRAPs: (*i*) We use 3 encoders per vRAP; (*ii*) We use one radio orchestrator per vRAP; and (*iii*) althoug we use a single CPU orchestrator, the number of neurons therein scales with the number of vRAPs as explained above. This complexity, however, is negligible during exploitation as the feed-forward neural networks comprising `vrAIn` involve simple matrix multiplications. For instance, computing an action takes less than 20ms for a cluster with 16 vRAPs on a single i7-8750 CPU core. Concerning training (back-propagation), one batch over 100 context-action pairs takes around 40ms in the same platform. Certainly, the use of GPUs would expedite these processes. Moreover, because we use a dedicated encoder/radio orchestrator per vRAP, the number of vRAPs do not affect their training speed. However, this complexity does have an impact on the convergence of the actor-critic used for the CPU orchestrator. We explicitly evaluate this in §5.2.

Training dataset.[14] To generate our pre-training set $\mathcal{D}$, we set up one vRAP and one UE transmitting traffic in different scenarios and repeat each experiment for both compute nodes (i7-5600U and i7-8650U), different Q parameters, and a wide set of control actions as shown in §1:

– *Scenario 1 (static).* The UE is located at a fixed distance from the vRAP and transmits Poisson-generated UDP traffic with fixed mean and fixed power for 60 seconds (i.e. three contextual snapshots). We repeat the experiment for different mean data rates such that the load relative to the maximum capacity of the vRAP is $\{1, 5, 10, 15, \ldots, 100\}$% and different transmission power values such that the mean SNR of each experiment is $\{10, 15, 20, \ldots, 40\}$ dB. Figs. 2, 3 and 4 visualize some results from this scenario.
– *Scenario 2 (dynamic).* We let the UE move at constant speed on a trajectory that departs from the vRAP location (maximum SNR), moves ∼25 meters away (minimum reachable SNR) and then goes back to the vRAP location. We repeat the experiment 12 times varying the speed such that the whole trajectory is done in $\{10, \ldots, 120\}$ seconds.
– *Scenario 3 (2 users).* We repeat Scenario 2 with two UEs moving in opposite directions, producing in this way patterns with different SNR variances.

14. Our dataset is available at https://github.com/agsaaved/vrain.

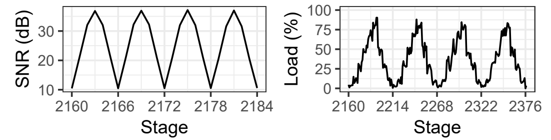

Fig. 12. Synthetic context patterns. SNR $\{\langle\bar{\sigma}_n^{(t)}\rangle, \langle\tilde{\sigma}_n^{(t)}\rangle\}$ patterns are generated by changing the UE's tx power to emulate one ∼120-s round-trip (Scenario 2 in §4.3) in 6 stages (left plot). Load $\langle\delta_n^{(t)}\rangle$ is sampled from a Poisson process with a mean that varies every 6 stages as $\bar{\delta} = \{5, 10, 30, 50, 70, 85, 50, 30, 10\}$% of the maximum capacity (right plot).

Digital twins. In order to pre-train `vrAIn` as explained in §3.4, we implement a digital replica for each vRAP i, each with a different combination of computing platform and Q, as we did to collect our training dataset $\mathcal{D}$. As explained in §3.4, we implement, for each vRAP i, two feed-forward neural networks $\tau_{i,\omega_1} \to (\tilde{J}_i)$ and $\tau_{i,\omega_2} \to (\tilde{\varepsilon}_i)$, each with 10 hidden layers and a total of 273 units and trained using a total of 8833 samples from our experience dataset (described above) and the Adam algorithm [39].

Offline batch training. We set the training batch size to $B = 100$ context samples, and generate 500 test contexts (not used in the training) for validation. We asses the performance of the test contexts every 200 episodes to evaluate the convergence in §5.2.

5 Learning Evaluation

Hereafter, we perform a thorough assessment of `vrAIn` that spans over §5, §6 and §7. To this aim, and unless otherwise stated, we use the synthetic context patterns shown in Fig. 12, with periods of 54 stages, for most of our online experiments. Note that these sequences are constructed to reflect extreme scenarios with high variability. We however evaluate `vrAIn` in a production RAN deployment in §7.

We start this evaluation, in this section, assessing `vrAIn`'s learning process. Specifically, we evaluate:

- The ability of `vrAIn`'s autoencoders to reduce the dimensionality of the input raw context sequences while preserving expressiveness (§5.1); and
- The convergence of `vrAIn` upon different settings, computing infrastructures and number of vRAPs (§5.2).

5.1 Encoder

The performance of `vrAIn`'s context encoders is essential to derive appropriate CPU and radio orchestration functions. We thus begin our evaluation by validating the design of our autoencoder.

First, we evaluate different encoder dimensions (ranging from 2 to 128) for the different sequence types of our context (mean SNR, SNR variance and data load patterns).

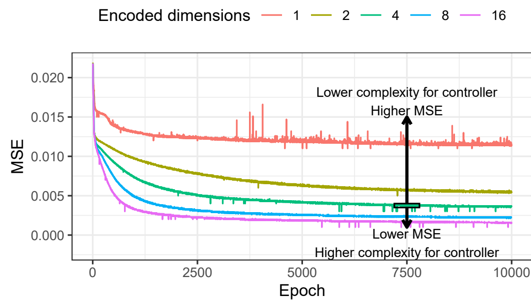

Fig. 13. Mean squared error (MSE) between validation dataset and reconstructed sequences for different number of encoded dimensions.

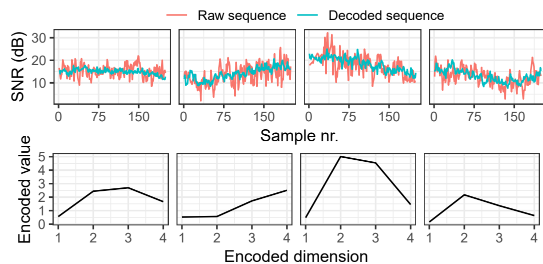

Fig. 14. Examples of 200-dimensional raw *vs* reconstructed SNR sequences (top). 4-dimensional encoded representations (bottom).

To this aim, we train our autoencoder with 66% of our pre-training dataset, leaving the remaining 34% for validation. Fig. 13 depicts the resulting mean squared error (MSE) of the reconstructed sequences for one sequence type (the mean SNR). From the figure, we conclude that 4 dimensions provide a good trade-off between low dimensionality and reconstruction error, and hence we use this value hereafter.

Second, we visually analyze if the encoder with the above setting captures higher-order pattern information. Fig. 14 shows a few examples of mean SNR sequences $\langle\bar{\sigma}^{(t)}\rangle$ from our pre-training dataset (red, top subplots) encoded into 4-dimensional vectors (bottom subplots) and reconstructed using the decoders introduced in §3.2.1 (blue line, top plots). We observe that the decoder reconstructs the input raw sequences remarkably well. We have observed a similar behavior for the other sequence types in our dataset: SNR variance $\langle\tilde{\sigma}^{(t)}\rangle$ and data load $\langle\delta^{(t)}\rangle$ (results omitted for space reasons). We hence conclude that our design is effective in projecting high-dimensional contexts into manageable representations—input signals of our controller.

5.2 Convergence

Offline learning scalability. We first assess the scalability of our offline training approach, introduced in §3.4. To this aim, we use Algorithm 2 to train `vrAIn` for different cluster sizes and computing capacities. To simplify the presentation, we focus on our i7-5600U computing infrastructure and set $Q = 7$ KBytes, but similar results are obtained for other settings. We hence set up vRANs with $\{1, 2, 4, 8, 16\}$ vRAPs and, respectively, $\{2, 4, 8, 12, 22\}$ cores of computing capacity; and plot in Fig. 15 the evolution over time of *normalized reward*—computed as $\frac{1}{P}\sum_{i\in\mathcal{P}}\tilde{J}_i^{(n)} - M\tilde{\varepsilon}_i^{(n)} - \lambda c_i^{(n)}$, where $\tilde{J}_i^{(n)}$ is the fraction of samples where the aggregate data queued by the vRAP is below a target Q_i and $\tilde{\varepsilon}_i^{(n)}$ corresponds to the fraction of unsuccessfully decoded subframes—CPU savings (relative to a static orchestration that provisions all computing capacity), and decoding error probability.

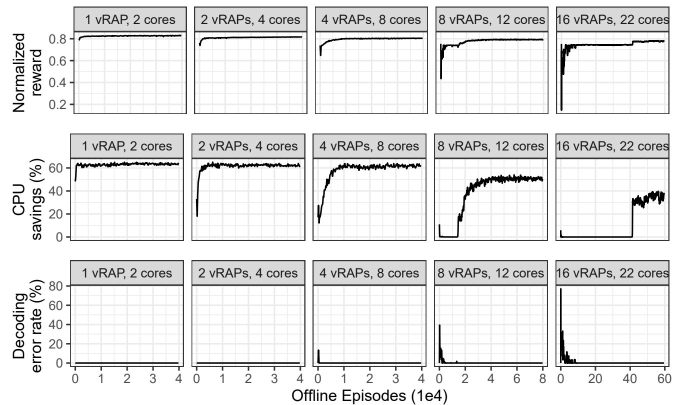

Fig. 15. Offline learning with different cluster sizes. Our offline training method introduced in §3.4 scales to clusters of multiple vRAPs.

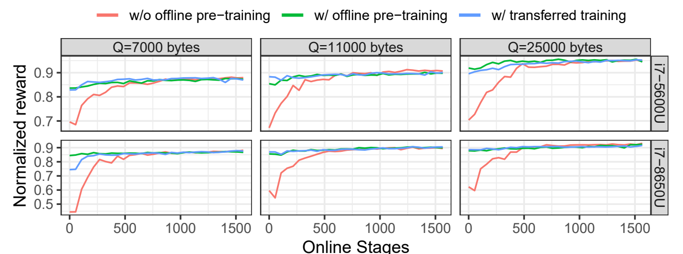

Fig. 16. Learning convergence of Algorithm 1 (online) with different settings. Pre-training offline with Algorithm 2 prior to online learning (Algorithm 1), even when using pre-training data from different platforms ("`w/ transferred training`"), expedites convergence.

For visualization purposes, we normalize reward between 0 and 1, where 0 corresponds to 100% buffer occupancy violation, 100% decoding errors and 100% CPU usage and 1 corresponds to 0% violation, 0% decoding errors and 0% CPU usage.

Unsurprisingly, larger clusters require longer convergence times because the space of contexts and actions grows exponentially with the number of vRAPs within the cluster. It is worth highlighting that this pre-training is done offline, as explained in §3.4, and can be expedited using specialized hardware such as GPUs and/or larger batch sizes. Hence, from these results, we conclude that `vrAIn` can easily handle clusters as large as 16 vRAP, which is well above the expected number of aggregated DUs for 5G [40], [41].

Online learning with different settings. We now evaluate the convergence of `vrAIn` during online operation, as introduced in Algorithm 1, i.e., interacting with our experimental vRAN platform. To this aim, we assess different configurations and computing infrastructures for a single-vRAP system with unconstrained computing capacity, to ease presentation. We hence present in Fig. 16 the evolution over time of normalized reward, as we did before.

We evaluate convergence for both computing infrastructures used in §4.3 and different values of Q, considering three training methods: (i) online training without offline pre-training, i.e., only Algorithm 1, labelled as "`w/o offline pre-training`"; (ii) pre-trained offline prior to online execution, i.e., Algorithm 2 prior to Algorithm 1, labelled as "`w/ offline pre-training`"; and (iii) pre-trained offline for a different computing platform, i.e., using Algorithm 2 for "i7-5600U" computing node prior to running `vrAIn` online (Algorithm 1) over "i7-8650U"

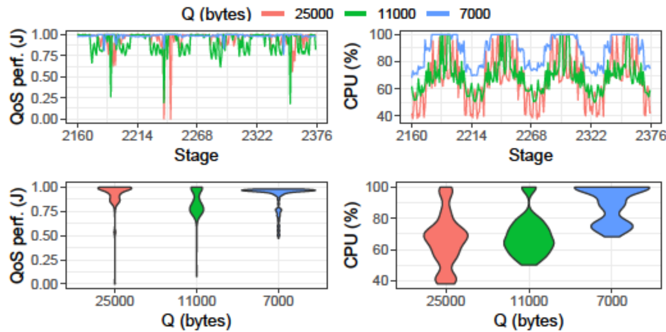

Fig. 17. Evolution over time (top) and distribution (bottom) of QoS performance (left) and CPU scheduling policy (right).

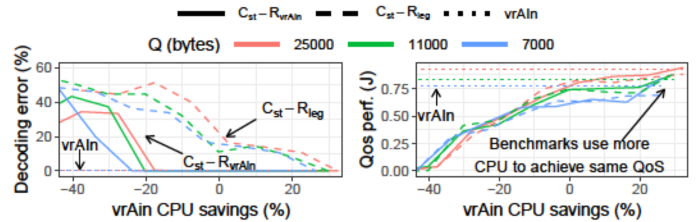

Fig. 18. `vrAIn` *vs* two benchmarks: $\mathtt{C}_{\mathtt{st}}$−$\mathtt{R}_{\mathtt{leg}}$ and $\mathtt{C}_{\mathtt{st}}$−$\mathtt{R}_{\mathtt{vrAin}}$. `vrAIn` renders a good trade-off between CPU allocations (cost) and QoS.

computing node (and *viceversa*), which we refer to as "`w/ transferred training`". As we can see from the figure, `vrAIn` requires between 500 and 1000 stages to converge for the highly dynamic contexts under evaluation *when it is not pre-trained*. As expected, when `vrAIn` is pre-trained offline, convergence becomes much faster. Furthermore, such offline pre-training does not necessarily have to be carried out from the same platform, as "`w/ transferred training`" allows fast convergence too, which allows reusing pre-training data across different vRAN systems.

6 Prototype Evaluation

Next, we evaluate our prototype implementation. To this aim, we use the same synthetic contexts shown in Fig. 12; and, to simplify the analysis, we focus hereafter our evaluation on our "i7-5600U" computing infrastructure only and assess `vrAIn` online in exploitation after offline training. Specifically, we assess the ability of `vrAIn` to:

- Achieve a good trade-off between cost (CPU usage) and QoS when there are sufficient CPU resources (§6.1); and
- Maximize performance and provision CPU resources efficiently across vRAPs when they are scarce (§6.2).

6.1 Unconstrained computational capacity

We first consider a single vRAP, which depicts a scenario where computational capacity is "unconstrained" since, as shown by Figs. 2-4, each of our vRAP prototypes requires one full CPU core at most.

Performance. We start our evaluation depicting in Fig. 17 (top) the temporal evolution of (*i*) QoS performance (J in eq. (2)), and (*ii*) compute control actions taken by `vrAIn`, for 4 periods (each comprised of 54 stages) of our synthetic context patterns, and the same Q values used before. Notice that `vrAIn` timely follows the context dynamic patterns shown in Fig. 12. In turn, Fig. 17 (bottom) presents the distribution across all stages. We draw three conclusions from these experiments. The *first conclusion* is that, the lower the parameter Q, the higher the CPU allocation chosen by `vrAIn`; indeed, higher CPU allocations induce lower decoding delay and thus lower buffer occupancy. The *second conclusion* is that higher Q targets render higher QoS performance, which is intuitive as lower Q implies requirements that are harder to meet. The *third conclusion* is that `vrAIn` achieves zero decoding error rate when not exploring. This is shown in Fig. 18 (left top) along with two benchmarks that we introduce next. We further observe that `vrAIn` follows load and SNR dynamics; also, as the computing capacity is always sufficient, the radio orchestrator does not bound the eligible MCSs (not shown for space reasons).

Static benchmarks. Let us consider two benchmarks:

(*i*) $\mathtt{C}_{\mathtt{st}}$−$\mathtt{R}_{\mathtt{leg}}$: Static CPU orchestration assigning fixed allocations and legacy radio orchestration (CPU-unaware);
(*ii*) $\mathtt{C}_{\mathtt{st}}$−$\mathtt{R}_{\mathtt{vrAin}}$: Static CPU orchestration and `vrAIn`'s radio orchestration ν, which is CPU-aware.

Note that $\mathtt{C}_{\mathtt{st}}$−$\mathtt{R}_{\mathtt{vrAin}}$ goes beyond the related literature closest to our work, which we revise in §8, namely [5], [8], as we augment such approaches with the ability to adapt the radio allocations to *both* SNR and traffic load dynamics. We apply the above benchmarks in our platform for the same contexts used before and for a wide range of static CPU assignments from $\{30, \ldots, 100\}$%. The results, shown in Fig. 18, depict the decoding error rate (left) and QoS performance (right) of both benchmarks as a function of `vrAIn`'s CPU savings for all static orchestrations (i.e., the mean saving/deficit that `vrAIn` has over the static orchestrations for the chosen CPU allocation). The results make evident the following points:

- Static CPU orchestrations that provide equal or less computing resources than `vrAIn`'s average allocation (**x-axis** $\leq$ **0**) render substantial performance degradation. Specifically, $\mathtt{C}_{\mathtt{st}}$−$\mathtt{R}_{\mathtt{leg}}$ yields high decoding error rate because it selects MCSs based on radio conditions only and does not take into account the availability of computing resources. Conversely, $\mathtt{C}_{\mathtt{st}}$−$\mathtt{R}_{\mathtt{vrAin}}$ worsens QoS performance because its CPU orchestrator fails to adapt to the context dynamics, e.g., data queues build up excessively during peak traffic;
- Static CPU orchestrations that increase the allocation of computing resources above `vrAIn`'s average allocation (**x-axis** $>$ **0**) only match `vrAIn`'s performance when the full pool of computing resources are allocated (with $>$ 20% more CPU usage over our approach).

As a result, *we conclude that `vrAIn` achieves a good balance between system cost and QoS performance.*

Dynamic benchmark. We next assess the performance of `vrAIn` in contrast to a simple dynamic CPU orchestration function that assumes a linear relationship between context, computing resources, and performance. This approach, which we refer to as $\mathrm{C}_{\mathrm{lin}}$-$\mathrm{R}_{\mathrm{vrAIn}}$, selects a computing control decision following $c_{lin}^{(n)} = C^{min} + \frac{\delta_n}{\delta^{max}}(C^{max} - C^{min})$, where C^{min} and C^{max} are, respectively, the minimum and the maximum values that the scheduling policy can take; δ_n denotes the new bit arrivals at stage n; and δ^{max} is the maximum value of bit arrival per stage. In this way, $\mathrm{C}_{\mathrm{lin}}$-$\mathrm{R}_{\mathrm{vrAIn}}$ selects a CPU scheduling policy within the interval $[C^{min}, C^{max}]$ proportionally to the normalized load δ_n/δ^{max}. Given the non-linear nature of our system, a linear approximation will fail to provision optimal results irrespective of the choice of $[C^{min}, C^{max}]$. Given that computing an

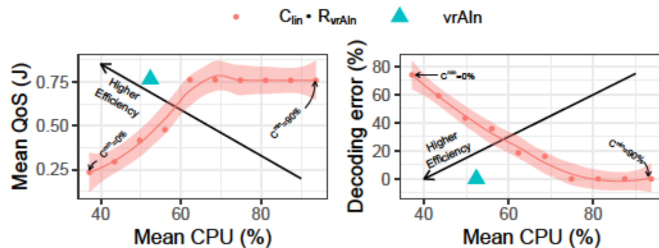

Fig. 19. vrAIn *vs* a simple dynamic computing orchestrator.

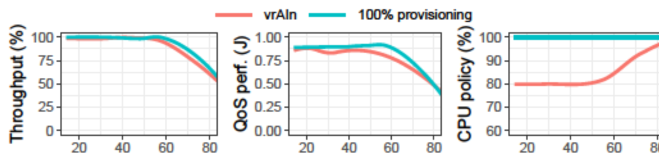

Fig. 20. Performance of vrAIn with heterogeneous UEs compared to a legacy computing-agnostic approach that uses all computing capacity.

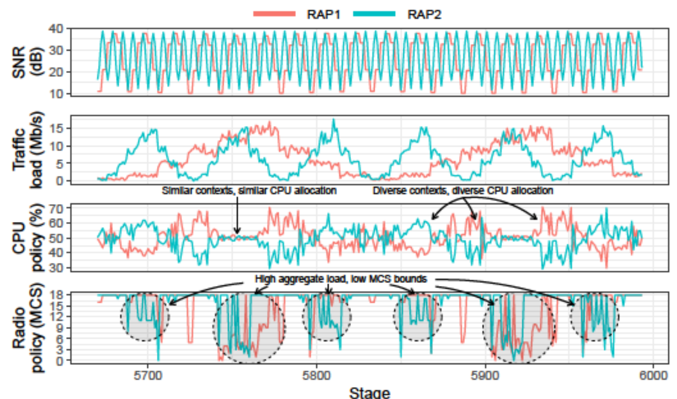

Fig. 21. vRAN with 2 vRAPs. vrAIn adapts the radio scheduling policy to minimize decoding errors (which are negligible and therefore not shown).

optimal C^{min} is hard and depends on the actual platform, we evaluate this benchmark for multiple values of C^{min}. Fig. 19, which shows the average CPU usage in the x-axis, and mean QoS performance and the decoding error rate, respectively, in the y-axis. We evaluate C_{lin}-R_{vrAIn} for different values of C^{min} ranging from 0% to 90% and compare the results with the performance attained by vrAIn. Note that, when $C^{min} = 0$, the selected CPU scheduling policy is very low for low traffic loads, providing a high rate of decoding errors. Conversely, high C^{min} values avoids decoding errors but yield over-provisioning of CPU resources. In contrast, vrAIn attains maximum QoS performance with roughly 10% less amount of CPU resources and 20% lower decoding error rate when compared to the best possible configuration of C_{lin}-R_{vrAIn}.

Heterogeneous UEs. By encoding SNR variance patterns $\tilde{\sigma}$ across all UEs in each vRAP, we enable vrAIn to adapt to contexts involving heterogeneous UEs. To analyze the behavior of vrAIn in such environments, we set up an experiment with two UEs (UE1 and UE2) attached to a vRAP. We fix the transmission power of UE1 such that its mean SNR is equal to 32 dB (high SNR) and vary the transmission power of UE2 to induce different values of SNR variance in the sequence of signals handled by the vRAP. To focus on the impact of the SNR variability, we fix the load of both UEs to 7.3 Mb/s and set $Q = 25$ Kbytes. Fig. 20 depicts the resulting aggregate throughput (relative to the load), QoS performance (J) and CPU scheduling policy when the SNR variance is $\tilde{\sigma} = \{15, \ldots, 80\}$, and use a scheduling policy that allocates all CPU resources to the vRAP as a benchmark ("100% provisioning"). We observe that throughput and J degrade as $\tilde{\sigma}$ increases, due to the lower signal quality of UE2. We conclude that vrAIn performs well under heterogeneous UEs, as it provides substantial savings over "100% provisioning" while delivering a similar performance.

6.2 Constrained computing capacity

To complete our experimental evaluation, we evaluate vrAIn under limited CPU capacity. To this end, we set up a second vRAP in our i7-5600U compute node limiting its capacity to a single CPU core, i.e., both vRAPs ("vRAP1" and "vRAP2") have to compete for these resources during peak periods. Moreover, we fix hereafter $Q_i = 7$ Kb $\forall i = \{1, 2\}$.

Analysis of vrAIn dynamics. We first let the vRAPs experience the same dynamic context patterns used before but *3 times slower* for "RAP1", i.e., each period of "RAP1" corresponds to 3 of "RAP2". This renders the SNR and load patterns shown in Fig. 21 (top) and allows us to study diverse aggregate load regimes. Fig. 21 depicts the temporal evolution of vrAIn's CPU scheduling policy (3rd plot) and radio scheduling policy (bottom plot). First, we can observe that vrAIn distributes the available CPU resources across both vRAPs following their context dynamics—equally between them when the contexts are similar. More importantly, we note that vrAIn reduces radio scheduling policies when the aggregate demand is particularly high to minimize decoding errors due to CPU deficit.

Comparison against benchmark approaches. We now assess the performance of vrAIn against the following benchmarks in scenarios with heterogeneous vRAPs:

(*i*) C_{vrAIn}-R_{leg}: vrAIn's CPU orchestrator and a legacy radio scheduler that is blind to the availability of CPU capacity.

(*ii*) R-Optimal: An oracle approach that *knows the future contexts* and selects the CPU and radio scheduling policies that maximize reward by performing an exhaustive search over all possible settings. Although unfeasible in practice, this provides an upper bound on performance.

(*iii*) T-Optimal: An oracle like R-Optimal that optimizes overall throughput instead of reward. Like R-Optimal, it is unfeasible in practice.

(*vi*) Heuristic: A linear model between MCS and CPU load is obtained by fitting a standard linear regression to our dataset. Using this model, we derive the CPU load needed by each RAP for the largest MCS allowed with the current mean SNR. If the system capacity is sufficient to handle such CPU load, we apply the resulting scheduling CPU/MCS policy. Otherwise, we apply the algorithm of [42] to obtain a fair CPU scheduling policy and use our linear model to find the corresponding MCS scheduling policy.

In order to evaluate these mechanisms, we use similar dynamic contexts to those of Fig. 21 but vary the average traffic load of "RAP2" $\bar{\delta}_2$ such that $\bar{\delta}_2 = k \cdot \bar{\delta}_1$ to illustrate the impact of heterogeneous conditions.

Fig. 22 shows the performance for all approaches in

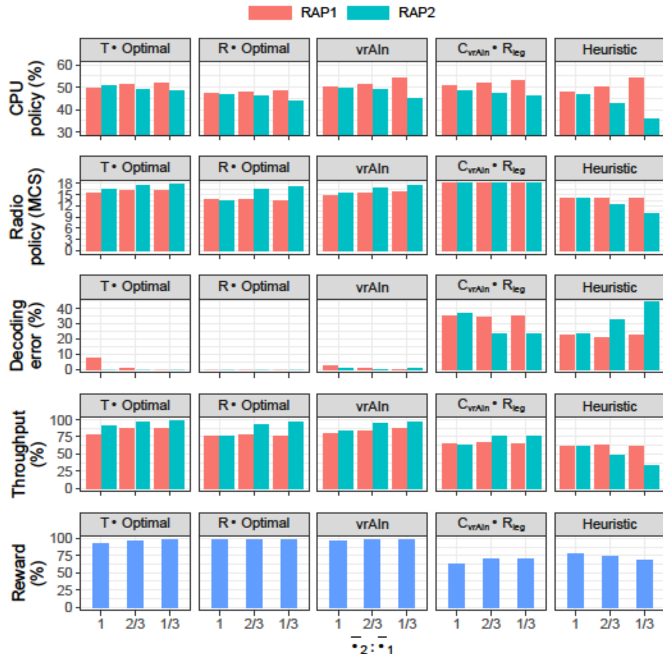

Fig. 22. vRAN with 2 heterogeneous vRAPs *vs.* 4 benchmarks: a throughput-optimal oracle (T-Optimal), a reward-optimal oracle (R-Optimal), vrAIn's CPU scheduling policy with a legacy radio scheduler blind to the CPU availability ($\mathrm{C_{vrAIn}}$-$\mathrm{R_{leg}}$), and an heuristic that leverages on a linear fit with our dataset.

terms of (i) CPU scheduling policy, (ii) radio scheduling policy, (iii) decoding error rate, (iv) throughput relative to the load, and (v) reward, for $k = \{\frac{1}{3}, \frac{2}{3}, 1\}$. The main conclusion that we draw from the above results is that vrAIn performs very closely to the optimal benchmarks (R-Optimal and T-Optimal) and substantially outperforms the other ones ($\mathrm{C_{vrAIn}}$-$\mathrm{R_{leg}}$ and Heuristic). Indeed, vrAIn provides almost the same reward as R-Optimal (the difference is below 2%) and almost the same throughput as R-Optimal (the difference is also below 2%). Furthermore, it provides improvements over 25% as compared to $\mathrm{C_{vrAIn}}$-$\mathrm{R_{leg}}$ and Heuristic both in terms of reward and throughput.

Looking more closely at the results for vrAIn, we observe that, as expected, the allocation of computing resources of our CPU orchestrator favors the RAP with higher load, i.e. "RAP1" for $k = \{\frac{1}{3}, \frac{2}{3}\}$, and provides very similar allocations for $\bar{\delta}_2 = \bar{\delta}_1$. In addition, we observe that vrAIn appropriately trades high MCS levels off for near-zero decoding error, selecting the highest possible MCS while avoiding decoding errors.

In contrast to vrAIn, $\mathrm{C_{vrAIn}}$-$\mathrm{R_{leg}}$ and Heuristic fail to select appropriate scheduling policies. The former fails to decode a large number of frames: *as it is blind to the computing capacity*, it employs overly high MCSs under situations of CPU deficit, and thus sacrifices roughly 25% of throughput *w.r.t.* vrAIn. The latter does adapt its radio scheduling policy to the CPU capacity; however, it does so employing an oversimplified model that does not provide a sufficiently good approximation and yields poor choices: in some cases, it selects overly high MCS bounds, leading to decoding errors, while in other cases it chooses overly low MCSs, leading to poor efficiency. As a result, Heuristic also sacrifices substantial throughput *w.r.t.* vrAIn (losing as much as 30% throughput).

TCP flows. Finally, we assess the performance of vrAIn with TCP traffic. Fig. 23 shows the performance of vrAIn using both TCP and UDP transport protocols for the same context dynamics used before when $\bar{\delta}_2 = \bar{\delta}_1$. The figure shows that both transport layer protocols obtain similar performance: vrAIn attains similar CPU savings for TCP and UDP (left plot of Fig. 23) without penalizing the overall throughput (right plot of Fig. 23). This shows that vrAIn works well under short-term traffic fluctuations such as the ones resulting from the adaptive rate algorithm of TCP.

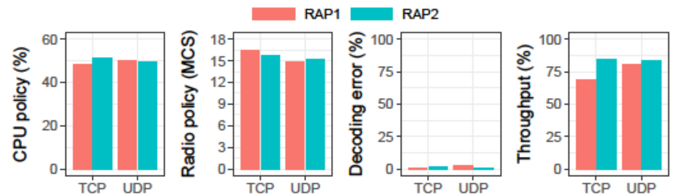

Fig. 23. vrAIn impact on TCP performance

7 Large-scale Evaluation

The above experiments do validate the feasibility of our approach in practice and highlight the main features of vrAIn when processing uplink traffic, the most demanding case in terms of computing resources. However, there are a number of issues that may appear when processing downlink traffic in large scale scenarios and deserve attention. In the following, we complement our prior experimental evaluation with a simulation-based study of the performance of vrAIn in a real-world RAN deployment and under different strategies for dimensioning computing capacity. To this aim, we now focus on downlink traffic and assume that each vRAP is managed by a single instance of vrAIn (each vRAN cluster contains one vRAP). Note that this is a *worst-case scenario for* vrAIn because we disable vrAIn for exploiting multi-vRAP diversity when pooling computing capacity, which provides substantial gains over legacy computing-agnostic schemes as demonstrated in §6.2. In this way, *the results shown below represent a lower bound of the gains attainable by* vrAIn.

7.1 Simulation framework

To assess the performance attained by vrAIn in a real-world scenario, we simulate vrAIn over a production RAN deployment in Romania[15], with 197 access points distributed as shown in Fig. 24a. As we can observe from the figure, there is a higher density of RAPs in the center (a big city) and the RAN is sparser by the outskirts (covering mostly highways and small commuter suburbs). In order to leverage our training data, and without loss in generality, we assume all RAPs are SISO 10-MHz LTE vRAPs with the same behavior as our LTE vRAPs analyzed in §1.

Our custom-built simulator follows the 3GPP guidelines for LTE performance evaluation [43] and its parameters are detailed in Table 2. The Signal-to-interference-plus-noise-ratio (SINR) perceived by the UEs is obtained by aggregating the interference of all active RAPs. For a given SINR, we compute the CQI ([16], Table III) of this UE, and then the maximum allowed MCS associated with this CQI according to 3GPP specification. Further, we implement a random mobility model for the UEs, ensuring a minimum

15. A statistical analysis of this network can be found in [15].

TABLE 2
Parameters of our simulation framework

Parameter	Value
System bandwidth	10 MHz
LTE subframe duration	1 ms
Transmission power	46 dBm
Antenna pattern	$A_H(\phi) = -min[12(\frac{\phi}{\phi_{3dB}})^2, A_m]$, $\phi_{3dB} = 70$ degrees $A_m = 25$ dB
Antenna gains	14 dBi
Path loss	128.1 + 37.6 · $\log_{10}(R[\text{Km}])$, R: vRAP to UE distance
Shadow fading	Lognormal distribution 10 dB standard deviation
Thermal noise	-176 dBm
Number of vRAPs	197
Mean Nr. UEs per vRAP	10
UE mobility	20% @ 100 km/h, 80% @ 3Km/h

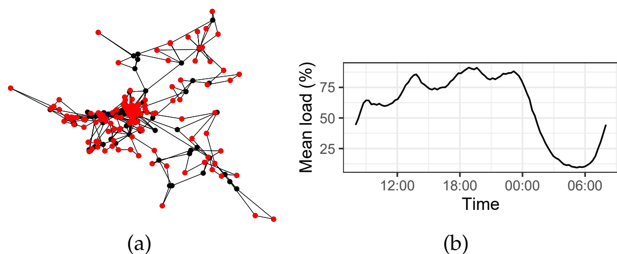

Fig. 24. (a) Deployment of an operational RAN in Romania. Red and black dots represent, respectively, radio sites and backhaul aggregation nodes. (b) Traffic load pattern over a period of 24h of a regular weekday.

distance between UE and RAP of 35 m, as recommended by 3GPP [43]. Finally, due to the difficulty to capture the computing behavior of our vRAPs in a tractable model for simulation (see §1), we have implemented a deep neural network (DNN), trained using our dataset, to determine when decoding errors occur due to lack of computing resources.

We simulate one regular week using synthetic traffic patterns obtained from [44], which emulate the behavior of real RANs at scale (see [44] for details). To simplify the analysis, in the following we focus on a 24-hour period during weekdays—with the traffic profile (relative to the capacity of the system) shown in Fig. 24b; but similar conclusions can be obtained from any other day. We further assume that the aggregate computing capacity of the whole vRAN is dimensioned to the minimum amount of CPU resources required such that no violations of the encoding deadlines due to CPU deficit occur during the load peak of the day when not using `vrAIn`, which we refer to as "`100% provisioning`". We also study the behavior of `vrAIn` when the system is under-provisioned to 70% and 85% of that computing capacity, which enables capital cost savings.

7.2 Performance and cost savings at scale

As demonstrated in our experimental campaign of §6, `vrAIn` provides substantial operational cost (OPEX) savings (reduction of CPU usage) when there is enough computing capacity (§6.1). We also demonstrated that `vrAIn` enables capital cost (CAPEX) savings too by adapting (maximize performance) to scenarios where computing capacity is under-provisioned (§6.2). Our goal here is to validate this behavior in a real-world deployment.

In this way, in the following, we analyze the throughput performance (relative to the capacity of the system),

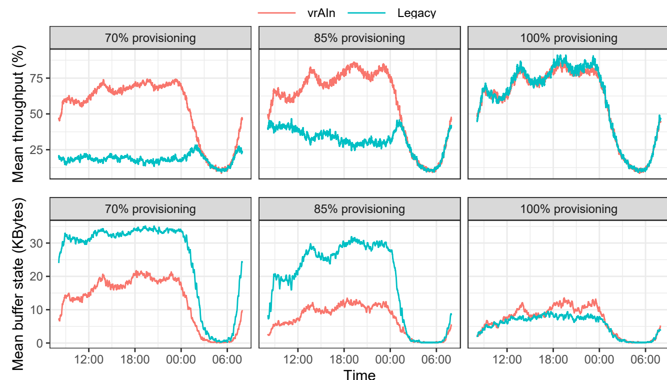

Fig. 25. Performance evolution over time for a computing capacity dimensioned for the peak load ("`100% provisioning`") and for 70% and 85% of that computing capacity ("`70% provisioning`" and "`85% provisioning`", respectively). Mean relative throughput (top). Mean buffer occupancy (bottom).

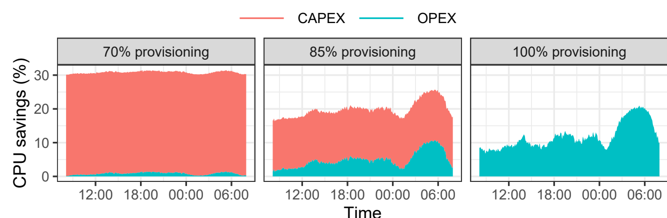

Fig. 26. `vrAIn`'s savings relative to a static computing-agnostic approach provisioned with sufficient CPU resources for the peak demand.

buffer state dynamics (a proxy of delay performance), CAPEX/OPEX savings and inter-cell interference with `vrAIn` and a legacy approach that uses all computing resources available and a legacy radio scheduler that is agnostic to the availability of computing resources.

Provisioning computing capacity for the peak. Let us first focus on the evolution of throughput performance, buffer states and cost savings when the computing capacity is provisioned to accommodate the peak load (right-most plots in Figs. 25 and 26, labeled as "`100% provisioning`"). From Fig. 25 we observe that `vrAIn` achieves roughly the same throughput performance and slightly higher buffer sizes (up to 5%) than "`Legacy`". This is explained because `vrAIn` trades off this slightly higher delay for substantial OPEX savings. This is shown in Fig. 26 (right-most plot, labeled as "`100% provisioning`"), where `vrAIn` achieves between 10% and 20% of OPEX savings. Note however that this difference in delay vanishes when `vrAIn` is configured to favor performance over OPEX savings as shown in §6.1 (results omitted here for the sake of space).

Under-provisioning of computing capacity. We now analyze the case where we impose under-provisioning to obtain aggressive CAPEX gains by reducing the availability of computing capacity to 85% and 70% relative to the dimensioning strategy discussed before, and labeled as "`85% provisioning`" and "`70% provisioning`", respectively, in Figs. 25 and 26. The evolution of throughput and buffer states in Fig. 25 for these two scenarios (left-most and middle plot, respectively) make it evident how `vrAIn` enables aggressive CAPEX savings while retaining high performance gains when compared to legacy computing-

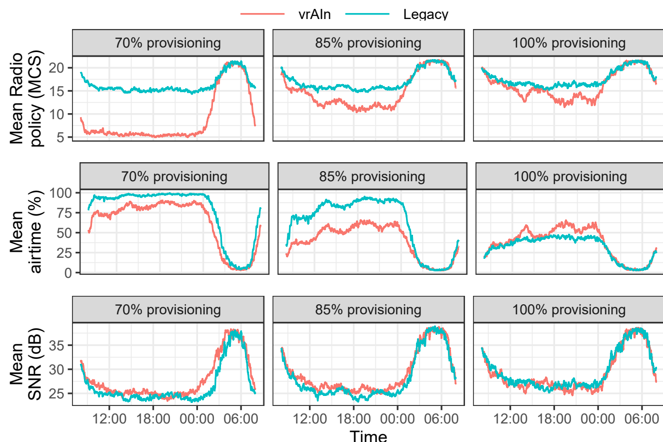

Fig. 27. Evolution over time of mean radio scheduling policy, vRAP airtime usage, SINR across all vRAPs.

agnostic approaches. Specifically, `vrAIn` provides up to 50% and 55% throughput gains over our legacy scheme when, respectively, the computing capacity is under-provisioned to 85% and 70% of the peak load, in average, and during the time of peak demand (between 18:00 and 00:00). The reason lies on the fact that `vrAIn`'s ability to optimize *jointly* radio *and* computing scheduling policies allows `vrAIn` to better accommodate the demand along the time domain, trading off delay for near-zero violations of the encoding deadlines due to a deficit of computing capacity. In contrast, legacy's agnostic behavior with respect to the availability of computing capacity yields substantial throughput loss due to a high rate of violations of the encoding deadlines during instantaneous peak demands. This produces a cascade effect causing large amounts of time wasted in re-transmissions and rendering even larger perceived latency: up to 160% and 73% higher buffer occupancy over `vrAIn` for 85% and 70% of computing provisioning, respectively, in average, and in the same period of peak demand.

A final remark is that `vrAIn` adapts, i.e., maximizes performance, to constrained computing environments. This can be observed by comparing the performance indicators of both "`70% provisioning`" and "`85% provisioning`" to those shown for "`100% provisioning`". In particular, `vrAIn` has no loss in throughput performance for "`85% provisioning`" and only up to 10% throughput loss for "`70% provisioning`", in average, and during the same period of peak demand discussed earlier. This contrasts with the substantial throughput loss attained by our legacy approach: up to 65% and 80% of throughput loss in those same conditions. In terms of latency, `vrAIn` only suffers from 1% increased in mean buffer occupancy in the case of "`85% provisioning`" relative to the buffer occupancy for "`100% provisioning`" (in contrast to the 282% increase of the legacy approach), and 73% increased in mean buffer occupancy with 75% under-provisioning (in contrast to the 337% increase of the legacy approach).

Inter-cell interference. The ability of `vrAIn` to trade-off delay for throughput maximization, particularly with under-provisioned computing systems, may, however, incur additional inter-cell interference. To assess this, we plot in Fig. 27 the evolution over time of the average SINR across all RAPs in the system (bottom plot), the relative channel airtime utilization averaged across all vRAPs (middle plot) and mean MCS index used for both `vrAIn` and a legacy approach. Perhaps surprisingly, we do not observe substantial difference between `vrAIn` and "`Legacy`" in terms of SINR, which leads us to conclude that `vrAIn` has negligible impact on inter-cell interference. The rationale is the following. As expected, `vrAIn`'s radio MCS patterns considerably change for different computing provisioning strategies. For instance, for 70% under-provisioning, we can observe that `vrAIn` uses considerably low MCSs, which yields remarkably high channel time utilization. In contrast, "`Legacy`" uses roughly the same mean MCS patterns irrespective of the availability of computing capacity as its choice of MCS only depends on the channel quality. However, because these choices render high rates of violations of the encoding deadlines (due to deficit of computing resources), substantial channel time is occupied with re-transmissions, as evidenced by the mean airtime patterns in the middle plot of Fig. 27. As a result, under-provisioning leads to an increased channel utilization for both `vrAIn` and "`Legacy`". Importantly, `vrAIn` *uses this additional channel time efficiently*, with useful transmissions, as opposed to legacy's wastage of time re-transmitting data that will not be able to be encoded in time due to deficit of computing capacity.

8 Related Work

There exists a large amount of literature on the management of wireless resources in cellular systems, namely, scheduling and MCS selection mechanisms, with different scenarios and optimization criteria (e.g., [45], [46], [47]). The advent of virtualized RAN has fostered some research work to understand the relationship between computing and wireless resources, e.g., [3], [5].

Theoretical work. The works of [8] and [5] set a theoretical basis for computationally-aware wireless control mechanisms. The former formulates a max-rate optimization problem subject to the availability of computational resources assuming the feasibility of real-valued rate allocation vectors. The later takes a step further and jointly optimizes MCS selection and physical resource block allocation, which results in a more practical approach. Nonetheless, both works rely on the same model relating computational requirements and SNR, which needs to be pre-calibrated for the platform and scenario they operate, assume full buffers (i.e. they study the resource allocation problem at the system's capacity boundary) and neglect variations on the arrival bit-rate load. This issue is addressed in [48], which combines real-time traffic classification and CPU scheduling in a mobile edge computing (MEC) setup. However, they also rely on a simplistic baseband processing model and a lack of experimental validation.

Experimental work. The work of [3] is one of the first experimental works that study the potential cost savings when exploiting the variations across LTE vRAPs' processing load. Nevertheless, the heuristic they propose does not consider variations on the SNR, which we have shown has a great impact on the overall system, and assume that the distribution of the load is known a priori. The authors of [49] implement `PRAN`, a programmable LTE vRAN system with dynamic computational resource sharing that however relies on a very simplistic resource demand prediction

model. PRAN's data-plane re-engineering is the most important contribution of [49], which is complementary to our work. The authors of [50] provide an experimental study on the relationship between throughput and computational demand, but it does not consider contextual dynamics (SNR, traffic load) and also relies on a very simplistic resource demand prediction model. RT-OPEX [6] implements a CPU scheduler tailored-made for vRAP workload, which can easily be integrated in vrAIn.

In contrast to this prior work, we make an in-depth experimental study of the relationship between performance, radio and computing resources. We conclude that the underlying model is far from trivial to design efficient optimization schemes as it depends on the context (SNR, traffic load patterns), the vRAP configuration and the computing platform. In light of this, our approach, vrAIn, exploits model-free learning methods to address the dynamic vRAN resource control problem and, as a result, adapts to contextual changes and/or different platforms.

9 Conclusions

Virtualized radio access networks (vRANs) are the future of mobile access design. In this paper, we have presented vrAIn, a resource orchestration solution tailored to vRANs that *dynamically learns* the optimal allocation of computing and radio resources. Given a specific QoS target, vrAIn determines the allocation of computing resources required to meet such target and, in case of limited capacity, it jointly optimizes radio configuration (MCS selection) and CPU allocation to maximize performance. To this end, vrAIn builds on deep reinforcement learning to adapt to the specific platform, vRAN stack, computing behavior, and radio characteristics. Moreover, we developed an offline pre-training method, which scales to clusters of multiple vRAPs, and relies upon past experience with the real system, observations artificially obtained with a digital twin and batch training to expedite learning convergence.

Our results shed light on the behavior of vrAIn across different scenarios, showing that vrAIn is able to meet the desired performance targets while minimizing CPU usage, and gracefully adapts to shortages of computing resources. We further showed that performance is close to optimal and vrAIn provides substantial gains over static assignments or simple heuristics. Moreover, when evaluated at scale using topological data from a real RAN, we showed that vrAIn can achieve large savings from both operational and capital infrastructure expenditures while maximizing performance.

To the best of our knowledge, this is the first work that thoroughly studies the computational behavior of vRAN, and vrAIn is the first *practical* approach to the allocation of computing and radio resources to vRANs, adapting to any platform by *learning* its behavior on the fly.

Acknowledgements

This work was partially supported by the European Commission through Grant No. 856709 (5Growth) and Grant No. 856950 (5G-TOURS); by Science Foundation Ireland (SFI) through Grant No. 17/CDA/4760; and AEI/FEDER through project AIM under Grant No. TEC2016-76465-C2-1-R. Furthermore, the work is closely related to the EU project DAEMON (Grant No. 101017109).

References

[1] A. Checko *et al.*, "Cloud RAN for Mobile Networks—A Technology Overview," *IEEE Communications Surveys Tutorials*, vol. 17, no. 1, pp. 405–426, Sep. 2015.

[2] V. Suryaprakash *et al.*, "Are Heterogeneous Cloud-Based Radio Access Networks Cost Effective?" *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 10, pp. 2239–2251, Oct. 2015.

[3] S. Bhaumik *et al.*, "CloudIQ: A Framework for Processing Base Stations in a Data Center," in *Proc. of ACM MobiCom 2012*, Istanbul, Turkey, Aug. 2012.

[4] P. Rost, I. Berberana, A. Maeder, H. Paul, V. Suryaprakash, M. Valenti, D. Wübben, A. Dekorsy, and G. Fettweis, "Benefits and challenges of virtualization in 5G radio access networks," *IEEE Communications Magazine*, vol. 53, no. 12, pp. 75–82, Dec. 2015.

[5] D. Bega, A. Banchs, M. Gramaglia, X. Costa-Perez, and P. Rost, "CARES: Computation-Aware Scheduling in Virtualized Radio Access Networks," *IEEE Transactions on Wireless Communications*, vol. 17, no. 12, pp. 7993–8006, Dec. 2018.

[6] K. C. Garikipati, K. Fawaz, and K. G. Shin, "RT-OPEX: Flexible Scheduling for Cloud-RAN Processing," in *Proc. of ACM CoNEXT 2016*, Irvine, USA, Dec. 2016.

[7] P. Rost *et al.*, "The Complexity-Rate Tradeoff of Centralized Radio Access Networks," *IEEE Transactions on Wireless Communications*, vol. 14, no. 11, pp. 6164–6176, Nov. 2015.

[8] P. Rost *et al.*, "Computationally Aware Sum-Rate Optimal Scheduling for Centralized Radio Access Networks," in *Proc. of IEEE GLOBECOM 2015*, San Diego, USA, Dec. 2015.

[9] J. A. Ayala-Romero, A. Garcia-Saavedra, M. Gramaglia, X. Costa-Perez, A. Banchs, and J. J. Alcaraz, "vrAIn: A Deep Learning Approach Tailoring Computing and Radio Resources in Virtualized RANs," in *Proc. of ACM MobiCom 2019*, 2019, pp. 1–16.

[10] J. Mendes *et al.*, "Cellular access multi-tenancy through small-cell virtualization and common RF front-end sharing," *Elsevier Computer Communications*, vol. 133, pp. 59–66, Jan. 2019.

[11] C. Marquez *et al.*, "How Should I Slice My Network?: A Multi-Service Empirical Evaluation of Resource Sharing Efficiency," in *Proc. of ACM MobiCom 2018*, Oct. 2018.

[12] J. X. Salvat *et al.*, "Overbooking Network Slices Through Yield-driven End-to-end Orchestration," in *Proceedings ACM CoNEXT 2018*, Heraklion, Greece, Dec. 2018.

[13] A. Garcia-Saavedra, J. X. Salvat, X. Li, and X. Costa-Perez, "WizHaul: On the Centralization Degree of Cloud RAN Next Generation Fronthaul," *IEEE Transactions on Mobile Computing*, vol. 17, no. 10, pp. 2452–2466, Oct. 2018.

[14] A. Garcia-Saavedra, X. Costa-Perez, D. J. Leith, and G. Iosifidis, "FluidRAN: Optimized vRAN/MEC Orchestration," in *Proc. of IEEE INFOCOM 2018*, Apr. 2018, pp. 2366–2374.

[15] A. Garcia-Saavedra, G. Iosifidis, X. Costa-Perez, and D. J. Leith, "Joint Optimization of Edge Computing Architectures and Radio Access Networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 11, pp. 2433–2443, Nov. 2018.

[16] M. T. Kawser *et al.*, "Downlink SNR to CQI Mapping for Different Multiple Antenna Techniques in LTE," *International Journal of Information and Electronics Engineering*, vol. 2, no. 5, pp. 757–760, Sep. 2012.

[17] C. Y. Yeoh *et al.*, "Performance study of LTE experimental testbed using OpenAirInterface," in *Proc. of ICACT 2016*, PyeongChang, Korea, Jan. 2016.

[18] D. Szczesny, A. Showk, S. Hessel, A. Bilgic, U. Hildebrand, and V. Frascolla, "Performance analysis of LTE protocol processing on an ARM based mobile platform," in *Proc. of 2009 International Symposium on System-on-Chip*, Oct. 2009.

[19] O-RAN Alliance, "O-RAN-WG1-O-RAN Architecture Description - v02.00.00." Technical Specification, Jul. 2020.

[20] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.

[21] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[22] O. Vinyals *et al.* (2019, Jan.) AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. [Online]. Available: https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/

[23] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: review, opportunities and challenges," *Briefings in Bioinformatics*, vol. 19, no. 6, pp. 1236–1246, Nov. 2018.

[24] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep Direct Reinforcement Learning for Financial Signal Representation and Trading," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 3, pp. 653–664, Mar. 2017.

[25] C. J. Watkins and P. Dayan, "Q-learning," *Springer Machine learning*, vol. 8, no. 3-4, pp. 279–292, May 1992.

[26] L. Tang, R. Rosales, A. Singh, and D. Agarwal, "Automatic Ad Format Selection via Contextual Bandits," in *Proc. of CIKM 2013*, San Francisco, USA, Oct. 2013. [Online]. Available: http://doi.acm.org/10.1145/2505515.2514700

[27] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, Aug. 2013.

[28] P. Turner, B. B. Rao, and N. Rao, "CPU bandwidth control for CFS," in *Proc. of OLS 2010*, vol. 10, 2010.

[29] F. Capozzi *et al.*, "Downlink Packet Scheduling in LTE Cellular Networks: Key Design Issues and a Survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 2, pp. 678–700, Jul. 2013.

[30] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *Proc. of ICLR 2016*, San Juan, Puerto Rico, May 2016.

[31] U. Demšar *et al.*, "Principal Component Analysis on Spatial Data: An Overview," *Routledge Annals of the Association of American Geographers*, vol. 103, no. 1, pp. 106–128, Jul. 2012.

[32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[33] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 518–529, Feb. 2015.

[34] S. Gu *et al.*, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Proc. of IEEE ICRA 2017*, Singapore, May 2017.

[35] C. Wang *et al.*, "Autonomous navigation of UAV in large-scale unknown complex environment with deep reinforcement learning," in *Proc. ofIEEE GlobalSIP 2017*, Montreal, Canada, Nov. 2017.

[36] D. Silver *et al.*, "Deterministic policy gradient algorithms," in *Proc. of ICML 2014*, Beijing, China, Jun. 2014.

[37] I. Gomez-Miguelez *et al.*, "srsLTE: An Open-source Platform for LTE Evolution and Experimentation," in *Proc. of ACM WiNTECH 2016*, New York City, USA, Oct. 2016.

[38] R. Mijumbi *et al.*, "Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262, Sep. 2015.

[39] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, Jan. 2017.

[40] Telefonica, Ericsson, "Cloud RAN Architecture for 5G," *White Paper*, 2016.

[41] L. M. Larsen *et al.*, "A survey of the functional splits proposed for 5g mobile crosshaul networks," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 146–172, 2018.

[42] C. Joe-Wong *et al.*, "Multiresource Allocation: Fairness-Efficiency Tradeoffs in a Unifying Framework," *IEEE/ACM Transactions on Networking*, vol. 21, no. 6, pp. 1785–1798, Dec. 2013.

[43] 3GPP TR 38.802, "Study on new radio access technology physical layer aspects (release 14)," 3rd Generation Partnership Project (3GPP), Tech. Rep., 2017.

[44] C. Marquez *et al.*, "Identifying Common Periodicities in Mobile Service Demands with Spectral Analysis," under review for MedComNet 2020. https://github.com/mgramagl/mywebsite/blob/master/static/files/MedComNet_2020.pdf.

[45] Y. Li, M. Sheng, X. Wang, Y. Zhang, and J. Wen, "Max-Min Energy-Efficient Power Allocation in Interference-Limited Wireless Networks," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 9, pp. 4321–4326, Sep. 2015.

[46] Z. Li, S. Guo, D. Zeng, A. Barnawi, and I. Stojmenovic, "Joint Resource Allocation for Max-Min Throughput in Multicell Networks," *IEEE Transactions on Vehicular Technology*, vol. 63, no. 9, pp. 4546–4559, Nov. 2014.

[47] M. Kalil, A. Shami, and A. Al-Dweik, "QoS-Aware Power-Efficient Scheduler for LTE Uplink," *IEEE Transactions on Mobile Computing*, vol. 14, no. 8, pp. 1672–1685, Aug. 2015.

[48] K. Wang *et al.*, "Computing aware scheduling in mobile edge computing system," *Springer Wireless Networks*, pp. 1–17, Jan. 2019.

[49] W. Wu *et al.*, "PRAN: Programmable Radio Access Networks," in *Proc. of ACM HotNets 2014*, Los Angeles, USA, Oct. 2014.

[50] T. X. Tran, A. Younis, and D. Pompili, "Understanding the Computational Requirements of Virtualized Baseband Units Using a Programmable Cloud Radio Access Network Testbed," in *Proc. of ICAC 2017*, Jul. 2017.

Jose A. Ayala-Romero received his M.Sc. and Ph.D. degrees from Technical University of Cartagena (UPCT), Spain, in 2015 and 2019, respectively. He was a visiting Ph.D. student with the DEI Department, University of Padova (2017), and NEC Laboratories Europe, Germany (2018). Since November 2019, he is a post-doctoral researcher with School of Computer Science and Statistics at Trinity College Dublin.

Andres Garcia-Saavedra received his PhD degree from the University Carlos III of Madrid (UC3M) in 2013. He then joined Trinity College Dublin (TCD), Ireland, as a research fellow. Since July 2015, he is a senior researcher with NEC Laboratories Europe. His research interests lie in the application of fundamental mathematics to real-life wireless communication systems.

Marco Gramaglia is a post-doc researcher at University Carlos III of Madrid (UC3M), where he received M.Sc (2009) and Ph.D (2012) degrees in Telematics Engineering. He held post-doctoral research positions at ISMB (Italy), the CNR-IEIIT (Italy) and IMDEA Networks (Spain). He was involved in EU projects and authored more than 40 papers appeared in international conference and journals.

Xavier Costa-Pérez (M'06–SM'18) is ICREA Research Professor, Scientific Director at the i2Cat Research Center and Head of 5G Networks R&D at NEC Laboratories Europe. He has served on the Organizing Committees of several conferences, published papers of high impact and holds tenths of granted patents. Xavier received his Ph.D. degree in Telecommunications from the Polytechnic University of Catalonia (UPC) in Barcelona and was the recipient of a national award for his Ph.D. thesis.

Albert Banchs received his M.Sc. and Ph.D. degrees from the UPC-BarcelonaTech in 1997 and 2002. He has a double affiliation as Professor at the University Carlos III of Madrid and Deputy Director of the IMDEA Networks institute. Before joining UC3M, he was at ICSI Berkeley in 1997, at Telefonica I+D in 1998, and at NEC Europe Ltd. from 1998 to 2003. Prof. Banchs has served in many TPCs and has also served in the editorial board of a number of journals, and is currently Editor of IEEE/ACM Transactions on Networking. Dr. Banchs has participated in many European projects and industry contracts, and is currently the Technical Manager Deputy of the European project 5G-TOURS.

Juan J. Alcaraz received his Ph.D. degree in telecommunications engineering from the Technical University of Cartagena (UPCT), Spain, in 2007. Between 1999 and 2004 he worked for several telecommunication companies until joining the UPCT in 2004, where he is currently an Associate Professor with the Department of Information and Communication Technologies. He was a Fulbright Visiting Scholar with the Electrical Engineering Department, University of California, Los Angeles (UCLA), in 2013, and a Visiting Professor with the Department of Information Engineering of the University of Padova in 2017 and in 2019.