Fair and Scalable Orchestration of Network and Compute Resources for Virtual Edge Services

Sharda Tripathi *Member, IEEE,* Corrado Puligheddu *Member, IEEE,* Somreeta Pramanik *Member, IEEE,* Andres Garcia-Saavedra *Member, IEEE,* Carla Fabiana Chiasserini *Fellow, IEEE*

Abstract—The combination of service virtualization and edge computing allows for low latency services, while keeping data storage and processing local. However, given the limited resource availability at the edge, a conflict in resource usage arises when both virtualized user applications and network functions need to be supported. Further, the concurrent resource request by user applications and network functions is often entangled, since the data generated by the former has to be transferred by the latter, and vice versa. In this paper, we first show through experimental tests the correlation between a video-based application and a vRAN. Then, owing to the complex involved dynamics, we develop a scalable reinforcement learning-based framework for resource orchestration at the edge, which leverages a Pareto analysis for provable fair and efficient decisions. We validate our framework, named VERA, through a real-time proof-of-concept implementation, which we also use to obtain datasets reporting real-world operational conditions and performance. Using such experimental datasets, we numerically demonstrate that VERA meets the KPI targets for over 96% of the observation period and performs similarly when executed in our real-time implementation, with KPI differences below 12.4%. Further, its scaling cost is 54% lower than a centralized framework based on deep-Q networks.

Index Terms—Virtual RAN, virtualized services, resource orchestration, machine learning, experimental testbed

1 Introduction

Network Function Virtualization (NFV) and edge computing are disrupting the way mobile services can be offered through mobile network infrastructure. Third parties such as vertical industries and over-the-top players can now partner up with mobile operators to reach directly their customers and deliver a plethora of services with substantially reduced latency and bandwidth consumption. Video streaming, gaming, virtual reality, safety services for connected vehicles, and IoT are all services that can benefit from the combination of NFV and edge computing: when implemented through virtual machines or containers in servers co-located with base stations (or nearby), they can enjoy low latency and jitter, while storing and processing data locally.

The combination of NFV, edge computing, and an efficient radio interface, e.g., O-RAN [1], is therefore a powerful means to offer mobile services with high quality of experience (QoE). However, some important aspects have been overlooked. On the one hand, user applications are not the only ones that can be virtualized: network services such as data radio transmission and reception are nowadays virtualized and implemented through Virtual Network Functions (VNFs) as well [2], [3], [4], [5], [6]; and both types of virtual services, user's and network's, may be highly computationally intensive. On the other hand, it is a fact that computational availability at the network edge is limited [7]. It follows that **in the edge ecosystem, user applications and network services compete for resources, hence designing automated and efficient resource orchestration mechanisms in the case of resource scarcity is critical.**

Further, looking more closely at the computational demand of virtualized user applications and at that of network service VNFs, one can notice that they certainly depend on the amount of data each service has to process, but they are also entangled [8]. As an example, consider a user application at the edge and (de-)modulation and (de-)coding functions in a virtualized radio access network (vRAN). For downlink traffic, the application bitrate determines the amount of data to be processed by the vRAN; on the contrary, for uplink traffic, the data processed by the vRAN is the input to the application service. A negative correlation, however, may also exist: the more data compression is performed by a user application, the higher its computational demand, but the smaller the amount of data to be transmitted and the less the computing resources required by the vRAN. In a nutshell, **a correlation exists between the amount of data processed/generated by virtual applications at the edge and network services VNFs, and such correlation can be positive or negative depending on the type of involved VNFs.**

Related joint resource problems have been addressed before [8] albeit ignoring the complex relationship between all system parameters and context variables and, therefore, making simplifying assumptions that do not work in prac-

S. Tripathi is with Birla Institute of Technology and Science Pilani, India; e-mail:sharda.tripathi@pilani.bits-pilani.ac.in. A. Garcia-Saavedra is with NEC Laboratories Europe, Spain; e-mail: andres.garcia.saavedra@neclab.eu. C. Puligheddu, S. Pramanik, and C. F. Chiasserini are with Politecnico di Torino, Italy; e-mail: firstname.lastname@polito.it. C. F. Chiasserini is also with IEIIT-CNR and with CNIT, Italy.

This work was supported partially by the European Commission through Grant No. 101095890 (Horizon Europe SNS JU PREDICT-6G project), Grant No. 101017109 (DAEMON project) and Grant No. 101095363 (Horizon Europe SNS JU ADROIT6G project), and partially by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART").

tice [9]. Our experimental analysis in Sec. 3 indeed unveils such complex couplings. For instance, contextual features of the wireless link, such as signal-to-noise-ratio (SNR), radio policies such as the modulation order and coding scheme (MCS) selected by the vRAN, and the computing resources allocated to the vRAN have non-linear effects on the resulting latency and, as a result, on the amount of buffering required by a video-based service. **These issues impair the use of modeling techniques traditionally used for optimal resource allocation in practical situations.**

To capture such trends and relations, and withstand the above challenges, we design a flexible and scalable framework, called VERA (Virtualized Edge for Radio and user Applications), leveraging a model-free reinforcement learning (RL) approach. Like us, other authors also use model-free approaches to address the aforementioned complex relationship across different aspects of a vRAN [10], [11]. A fairly common solution technique is based on the use of deep-Q networks (DQN). Several resource allocation problems in computationally constrained environments [12], [13], [14], [15] and other related issues like joint server selection, task offloading, and handover [16], [17], [18], [19], [20], [21], [22], [23] in multi-access edge computing wireless networks have been tackled through DQNs. However, extending such approaches to a multi-service scenario falls into serious scalability issues. To address this problem, we adopt a distributed multi-agent learning approach. Not surprisingly, designing a multi-agent learning framework where the actions of individual agents must collectively satisfy the hard capacity constraints characteristic of mobile edge platforms is inherently hard. Inspired by [24] and other literature on autonomous driving, we decompose the policy function into two stages; the first stage produces *greedy* actions based on the context collected from the environment while the latter refines these actions to enforce hard constraints. Unlike previous work in other settings, however, our use case requires some notion of fairness when enforcing these constraints. To address this, we design a novel Pareto component that guarantees a fair Pareto-efficient solution.

In summary, we provide the following contributions:

- First, we present *experimental evidence* for the above observations, through a containerised edge and a software-defined-radio (SDR)-based vRAN testbed, which not only forms the basis of the design of the proposed VERA framework, but also provides a new perspective to the problem of deploying network services and user applications at the edge;
- Then, given the many interplaying factors and their complex interaction, *we introduce an RL model* for an effective, joint allocation of computing resources for user applications and vRAN at the edge;
- To aid scalability, we resort to *distributed learning agents*, which we complement with a Pareto analysis for a fair and efficient decision-making, whenever resource utilization is constrained to a given budget;
- We show the excellent performance of the VERA framework in terms of convergence as well as its ability to closely meet the target KPIs of all services in resource-constraint scenarios. Specifically, we show that, post convergence, VERA meets the KPI targets for more than 96% of the observation period, and that it performs similarly when executed in our real-time proof-of-concept implementation, with KPI differences below 12.4%. Further, we remark that the scaling cost of VERA is 54% lower compared to a competitive centralized framework using DQNs.
- Finally, we validate our approach by implementing VERA on our testbed and showing that its performance is preserved when interacts with a real system in real-time.

We remark that, to our knowledge, we are the first to address the allocation of a **common pool of edge resources to different**, competing, virtualized services through distributed learning, and to tackle the non-trivial correlations existing among the behaviors of such services *in a scalable manner*. Thanks to the distributed learning adopted in VERA, its action space is limited (as we show in Sec. 6) thereby providing VERA an edge over its contemporary techniques like DQNs in terms of scalability and quality of learning. Moreover, not only VERA can swiftly adapt to time-varying network conditions and application traffic, but it also controls the settings of both user applications and vRAN, selecting at each decision step a fair Pareto-efficient solution.

The rest of the paper is organized as follows. Sec. 2 introduces the reference scenario and system architecture. Our experimental analysis is presented in Sec. 3, highlighting the relevant components of the environment contextual information, the target KPIs, and the driving factors determining the system behavior. Sec. 4 describes the VERA framework, Sec. 5 presents our proof-of-concept and testbed. Sec. 6 shows VERA's performance, compares it against a state-of-the art alternative, and validates our approach by running the complete framework on our testbed. Finally, Sec. 7 discusses related work, and Sec. 8 draws our conclusions.

2 Reference Scenario and System Architecture

The system architecture and reference scenario under study are depicted in Fig. 1. For clarity, we focus on one type of vertical service and one virtual base station (vBS) – labelled as Radio Unit (RU) in the figure, – implemented within an edge computing platform. Note that, as explained in Sec. 4, VERA is highly scalable and can effectively handle multiple coexisting services as well as multiple coexisting vBSs (and/or network slices therein).

As sample use case, we consider a livecast service representing a live video recording of an event occurring, e.g., at a stadium, that is broadcast to multiple mobile users located therein or in the nearby area. The high-quality source video is processed within an edge computing platform through a standard video transcoding service. We remark that deploying services like livecast at the edge entails that video traffic is produced and consumed locally, thus saving network bandwidth. Further, it can leverage a multi-access edge computing (MEC) platform and the associated Radio Network Information Service (RNIS) [25], which, through feedback-based multicast, allows the use of estimated radio

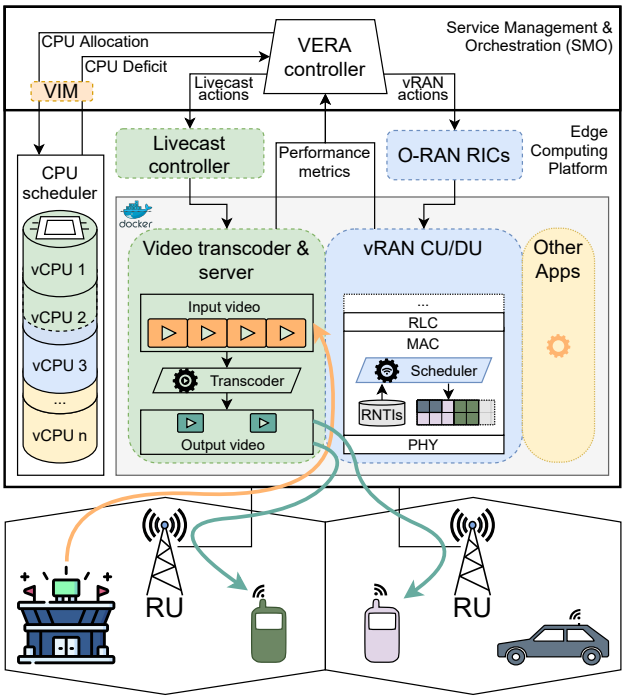


Fig. 1. Virtualized user application and vRAN at the edge: system scenario and reference use case

channel conditions for real-time tuning of the video coding parameters [26]. Finally, emerging interactive services for which video streaming is one of the essential components, e.g., crowdcast, augmented reality and mobile gaming, have such strict latency constraints that only an edge-based architecture can meet.

In addition to the livecast service (as well as, possibly, other user services running at the edge), the edge computing platform hosts vBS functions, central unit (CU) and/or distributed unit (DU), which are jointly controlled by the VERA controller. As depicted in Fig. 1, the VERA controller is deployed in the Service Management & Orchestration (SMO) platform, and interacts with both O-RAN intelligent controllers (RIC) to configure the vBS functions, the edge service controllers (in this case the livecast controller), and the NFV virtual infrastructure manager (VIM) to configure the CPU schedulers (see O-RAN specification [1]). In this way, VERA's workflows (data collection and decision making) are fully compliant with O-RAN's machine learning procedures [1]. Indeed, VERA continuously monitors the state of the vRAN and the livecast application (hereinafter also referred to as *services*), as well as the overall usage of computing resources in the edge platform. Then, it uses such observations to compute the values of the operating parameters for both livecast and vRAN, which, given the available computing and networking resources, meet both the application and vRAN KPI targets.

3 Experimental Analysis

The system architecture described in Sec. 2 has been recreated in a smaller scale in our testbed for the development and testing of VERA. The main components are the edge computing platform, and the user equipments (UEs), which communicate by means of an LTE vRAN implemented using the srsRAN suite [27]. The edge platform runs two Docker containers implementing, respectively, the livecast and the vRAN service, which consume the edge resource pool.

Our experimental vRAN testbed includes one srseNB instance, i.e., the LTE vBS, and two srsUE instances, which represent the recipients of the video content livecast by the vBS. The livecast application consists of a live video streaming transcoder, implemented with ffmpeg[1], and a server, based on ffserver[2]. It receives the high-quality original video and transcodes it through the recent VP9 codec using the bit rate and frame rate settings provided by VERA. Then, the transcoded video is served to the UEs, each running a player that streams, decodes, and plays the video. The radio and livecast services are connected to VERA through a dedicated API, used to dynamically set radio and livecast operating parameters and retrieve performance measurements. VERA also interacts with the edge computing platform operating system and the Docker daemon to monitor and allocate computing resources to the services. More details about our testbed are provided in Sec. 5.2.

We then use our testbed to analyze empirically the trade-offs between different actions configurable in our system, given different contexts.[3] To ease the analysis, we focus on a single user, but we note that VERA supports multiple users and we evaluate VERA with multiple users in later sections.

We start by defining a contextual feature that characterizes the videos being delivered by the livecast service:

- **Context 1:** Video input bit rate, input frame-per-second (FPS) rate, and input resolution;

and another feature that indicates the computational demand required by the livecast service:

- **Context 2:** Video CPU throttled time. This feature gives us an indication of the processing pressure associated with the requested video, which is a footprint distinguishable across videos, and hence impacts the overall performance and the choice of appropriate actions.

We also define three sets of actions for our livecast service, some of which re-encode each video accordingly:

- **Action 1:** Video output bit rate;
- **Action 2:** Video output FPS rate; and
- **Action 3:** CPU resources allocated to the livecast service;

and a KPI that we can use to estimate the quality of the video being delivered, as set forth below:

- **KPI 1:** Weighted Video Multimethod Assessment Fusion (WVMAF). It is based on the VMAF, a widely used objective metric to assess video quality, which provides a score between 0 (worst) and 100 (best) *per video frame.* The score is computed by aggregating different components such as Visual Information Fidelity, Detail Loss Metric, or Mean Co-Located Pixel

1. https://ffmpeg.org/
2. https://trac.ffmpeg.org/wiki/ffserver
3. The dataset will be made publicly available upon paper publication.

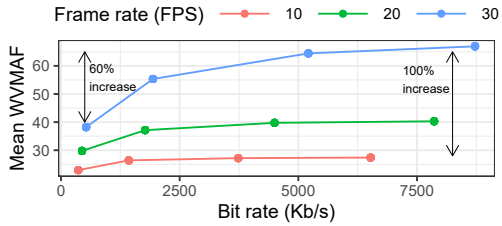


Fig. 2. Video quality (WVMAF) for different livecast service configurations (output FPS rate and bit rate). WVMAF increases by 100% by changing the output FPS from 10 to 30, and by 60% when increasing the output bit rate from 0.5 to 8 Mb/s

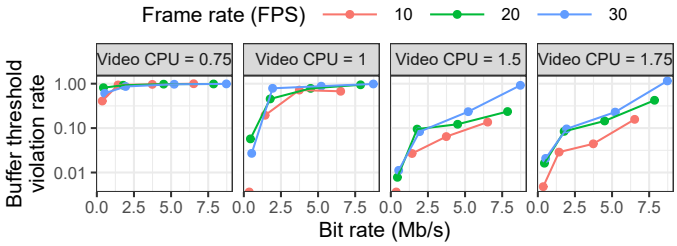


Fig. 3. Buffering threshold violation rate for different livecast service configurations and CPU allocations

Difference (the interested reader can find further details in [28]). However, because VMAF assesses the quality of individual video frames only, it is not helpful to measure the *smoothness* of a video, which is well known to impact the perceived quality. To weight this in, we amplify or attenuate the measured VMAF of each frame by the ratio between the output frame rate and the input frame rate, and we refer to this metric as WVMAF.

Fig. 2 shows the mean WVMAF score for a wide variety of VP9-encoded videos with different output FPS rates and bit rates, CPU throttle time, and (for simplicity) the same resolution. The results are intuitive: higher-bit-rate and higher-FPS videos have higher WVMAF. It is interesting to observe from Fig. 2 that the increase in mean WVMAF is 100% when the output FPS changes from 10 to 30, while it is just 60% for FPS 30 when the output bit rate increases from 0.5 to 8 Mbps. Thus, the frame rate setting has a larger impact on WVMAF than the target bit rate, which moreover shows diminishing returns. This is due to the weight that amplifies the measured VMAF, which increases the score when higher frame rates are used.

A second relevant KPI to assess the perceived QoE of the livecast service is:

- **KPI 2:** Video player buffering. Information about the client's buffer state, which stores video frames for playout, is a good estimator of the user's QoE [29]. Specifically, when the buffer size gets close to zero, the user's player may stutter, which resorts in a low level of QoE.

To assess this KPI, we select a threshold equal to 0.5 seconds of video buffered at the client's video player, and report in Fig. 3 the frequency that such threshold is violated for the same set of videos used before and for different combinations of actions. In this case, the target bit rate and the amount of CPU resources, measured in units of virtual CPU (vCPU) assigned to the service, have a much larger impact on this KPI than before (ruling ranges of this KPI that span 2 orders of magnitude). Note the logarithmic scale in the y-axis, which indicates a non-linear behavior.

Next, we focus on the vRAN service, and define two more actions related to it:

- **Action 4:** CPU resources allocated to the radio; and
- **Action 5:** A Modulation and Codign Scheme (MCS) policy. This policy follows that used in [9] and imposes an upper bound on the MCS eligible by the base station, which helps to control the computational demand of the radio service;
- **Action 6:** Bandwidth allocated to each UE, measured as the aggregate amount of radio Resource Blocks (RBs);

and three relevant contextual features:

- **Context 3:** Radio CPU throttled time, which is the radio counterpart of Context 2;
- **Context 4:** Signal-to-noise-ratio (SNR), which is a common feature used to estimate the quality of a wireless channel and in turn bounds its capacity; and
- **Context 5:** Network load, which corresponds to the offered load that the vBS has to process, generated by the applications deployed at the edge (such as our livecast service) and background data related to the mobile network.

Concerning radio KPIs, we first define:

- **KPI 3:** Radio latency. This is the latency associated with the data transmitted successfully over the air.

To analyze this KPI, we plot in Fig. 4 every data frame that violates/meets a latency threshold equal to 150 ms with red/blue colored dots when the vBS has to deliver randomly chosen videos from our set. We present these as functions of two of the radio contextual features (SNR and network load) and for different combinations of actions. Correlations between context, actions, and latency are evident. For instance, a higher MCS policy show a consistent improvement in latency performance, which however requires more computing resources. We observe a similar behavior when allocating a higher number of RBs (results omitted to reduce clutter).

We then define one last KPI associated with the radio service:

- **KPI 4:** Packet loss rate, which measures the number of unacknowledged TCP segments due to corruption on the radio link.

Fig. 5 shows this KPI for all the videos in our set as a function of the SNR (Context 4) and for different MCS policies (Action 5). In red, we mark those samples that exceed 1% packet loss. Obviously, the sample set is highly biased towards 0 packet loss rate. However, there are a number of video scenes that cause packet losses, and these are highly correlated with SNR and our MCS policy in a non-trivial manner as shown by the plot. For instance, we can observe that higher MCS policies yield considerably lower packet loss rate. Note that this gain comes from the extra wireless capacity granted by larger MCS policies and

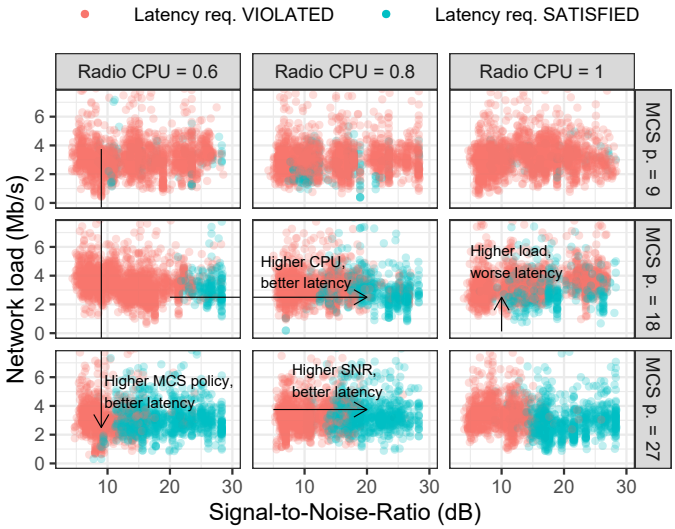


Fig. 4. Latency violations for different radio service configurations, contexts, and CPU allocations

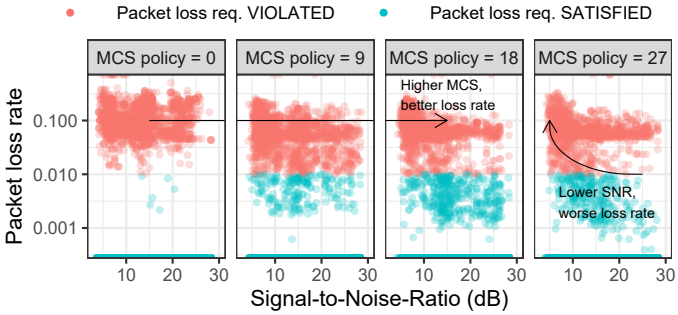


Fig. 5. Packet loss rate for different vRAN configurations and contexts

not from the reliability of a given MCS that is selected automatically by the radio scheduler: we simply impose a restriction on the set of eligible MCSs. Moreover, better SNR provides also better performance, which is intuitive. However, this relationship is non linear (note the logarithmic y axis). Likewise, the dependency between packet loss and the number of allocated RBs is also monotonic, i.e., high packet losses are observed with overly low RB allocations (like before, we omit these results to be concise).

It may be noted that we do not consider throughput, another commonly used metric, as a radio KPI in the VERA framework, since for delivering a real-time network load across a vRAN, packet loss and latency are more relevant metrics [30]. Nonetheless, it is implicit that as long as observed packet loss and observed latency values are as desired (which we later quantify in terms of KPI targets), the system throughput will be maximized since no packets are lost and all the traffic belonging to all the services is served in due time.

To conclude our experimental analysis, we plot in Fig. 6 a livecast KPI (buffer state) as a function of SNR (radio context) and MCS policy (radio action). Evidently, the buffer dynamics of the client's video player are highly correlated with the context and the actions performed over the livecast service. This proves that the resource orchestration problem we endeavour into in this paper is a coupled problem and all these edge services must be optimized jointly.

Conclusion: It becomes evident that the support of different applications in the edge platform leads to complex inter-dependencies between system parameters, context-action variables and KPIs, thereby making optimum resource allocation a challenging task. To this end, we propose the VERA framework, which is completely data-driven, and, hence, a good match for flexible and effective decision making in virtualized environments, despite the system complexity.

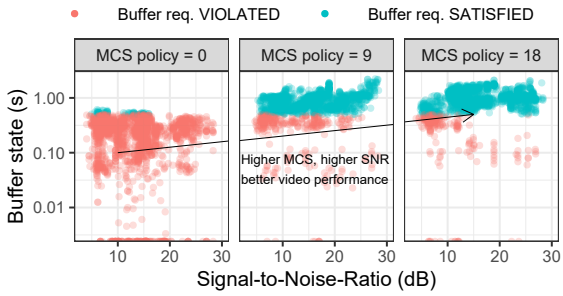


Fig. 6. Livecast performance (buffer state) for different vRAN configurations and contexts

4 The VERA Framework

The VERA framework is designed using a model-free RL approach. It includes distributed learning agents, each corresponding to a service in the edge platform, which simultaneously make decisions for the allocation of radio and computing resources as well as tune service-specific operating parameters. This design choice is key to attain *a scalable solution*. These decisions are hereafter collectively referred to as a resource allocation policy, which consists of two development stages:

- In the first stage, each RL agent makes decisions based on the shared context representation to obtain a greedy resource allocation policy;
- In the second one, greedy policies from all RL agents are collated and further refined in view of the feasibility of the chosen actions to obtain a Pareto-efficient fair resource allocation policy.

The structure of the VERA framework is shown in Fig. 7. Decisions are made with periodicity equal to $N \geq 1$ monitoring slots, i.e., an action is selected at the end of every decision window of duration N slots, and it is applicable to the subsequent N monitoring slots. The individual stages are elaborated in the sequel.

4.1 Notation

$\mathbb{R}^n$ denotes the set of n-dimensional real vectors. Vectors (usually in column form) are written in bold font, matrices are in upper-case, bold font, and sets are in calligraphic font. Subscripts and superscripts denote an element in a vector and elements in a sequence (resp.). E.g., $\langle \boldsymbol{x}^{(t)} \rangle$ is a sequence of vectors with $\boldsymbol{x}^{(t)} = [\boldsymbol{x}_1^{(t)}, \ldots, \boldsymbol{x}_n^{(t)}]^\top$ (superscript $\top$ is the transpose operator). $\boldsymbol{x}_i^{(t)}$ is the i-th component of the t-th vector in the sequence.

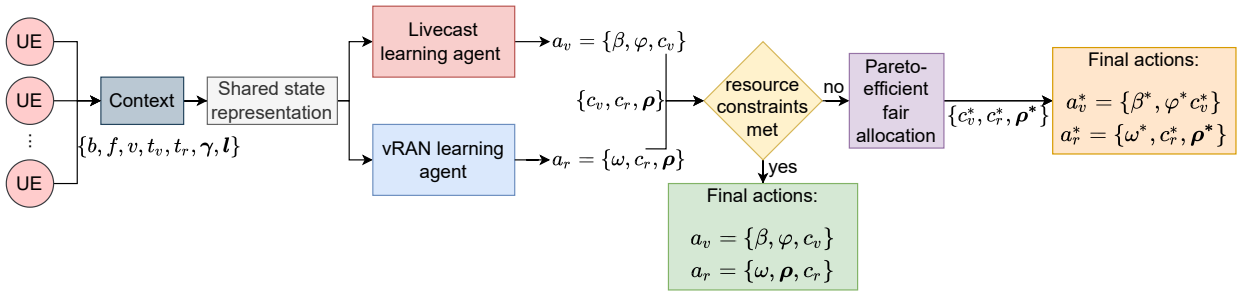


Fig. 7. Structure of the VERA framework

TABLE 1
Notation

Symbol	Description
Context notation	
b	Video input encoding bit rate
f	Video input FPS rate
v	Video input resolution
t_v, t_r	Normalized video and radio (resp.) throttled time
$\boldsymbol{\gamma}$	CQI value
$\boldsymbol{l}$	Livecast network load
Action notation	
β	Video output encoding bit rate
φ	Video output FPS rate
$\boldsymbol{c}$	CPU allocation
ω	vRAN MCS policy
$\boldsymbol{\rho}$	RB allocation
KPI notation	
ζ	WVMAF score
σ	Client's buffer state
λ	Latency
μ	Packet loss rate
$\{\zeta_o^m, \sigma_o^m, \lambda_o^m, \mu_o^m\}$	Observed KPI values by m-th UE
$\{\zeta_t, \sigma_t, \lambda_t, \mu_t\}$	Target KPI values

4.2 Greedy analysis

Since the edge platform may have several services consuming the resource pool, owing to resource sharing, the KPI satisfaction of each is interdependent. We therefore collect input variables pertinent to different services to form a context vector. The context vector is processed through an autoencoder to create a shared context representation that captures the correlation among context variables, as well as reduces the dimensionality of the context vector. Then, each RL agent devises a greedy resource allocation policy by using the same shared context representation and by mapping it onto an action vector such that its long-term cumulative reward from the environment is maximized. Notice that, although the decisions are based on shared context, greedy policies do not ensure that the sum of capacity-constrained resources among all the services does not exceed their maximum capacity. *To solve this issue, we design a Pareto algorithm that allows for feasible and fair resource sharing*. The elements composing the greedy resource allocation policy are introduced below, while the notation we use is summarized in Tab. 1.

Context space. As described in Sec. 3, the resource allocation for the livecast service is governed by the following contextual information: input bit rate (b), input video FPS (f), and input resolution (v) of the streaming video (i.e., Context 1 in Sec. 3). Besides, to accommodate any backlog in video processing, the normalized CPU throttled time of the livecast application (t_v) in the previous monitoring slot is considered (Context 2).

Likewise, resource allocation for the vRAN is based on normalized CPU throttled time (t_r) (Context 3), the 3GPP-compliant Channel Quality Indicator (CQI) ($\boldsymbol{\gamma}$) reported from UEs to vBS, which is representative of the SNR (Context 4), and the traffic from the livecast application sent over the radio link to the UEs (Context 5), specified by the network load ($\boldsymbol{l}$). Thus, the context vector observed in monitoring slot n ($n = 1, \ldots, N$) can be written as $\boldsymbol{x}^{(n)} \in \mathcal{X}$, $\boldsymbol{x}^{(n)} := \{b^{(n)}, f^{(n)}, v^{(n)}, t_v{}^{(n)}, t_r{}^{(n)}, \gamma_1^{(n)}, \cdots \gamma_M^{(n)}, l_1^{(n)}, \cdots l_M^{(n)}\}$.

Further, to extract the correlation between context variables, an autoencoder projects context vector $\boldsymbol{x}^{(n)} \in \mathcal{X}$ onto its latent representation $\boldsymbol{y}^{(n)} \in \mathbb{R}^D$, $\boldsymbol{y}^{(n)} := \{\boldsymbol{y}_1{}^{(n)}, \ldots, \boldsymbol{y}_D^{(n)}\}$ where $D < \dim(\mathcal{X})$. The latent representation $\boldsymbol{y}^{(n)}$ is shared with each RL agent so that its decision process for a given service is informed of the performance of others accessing the resource pool, thus representing a shared context representation. The autoencoder is implemented through a simple feed forward neural network that is activated using rectified linear units in the hidden layers. Note that dimensionality reduction is only one advantage of the autoencoder: indeed, it is primarily used to capture multimodal patterns among context variables, which may not otherwise be evident owing to the system complexity.

Action space. Since services are heterogeneous, we define action space $\mathcal{A} := \{\boldsymbol{a}_k\}$, $\forall k \in (1, \ldots, K)$, comprising action vectors each having service-specific action variables. In our reference scenario, $K = 2$, and we associate $k = 1, 2$, respectively, to action vectors for livecast and vRAN. Consequently, $\boldsymbol{a}_1$ comprises the CPU allocated to the livecast application (c_v), i.e., Action 3 in Sec. 3, the video output encoding bitrate (β), i.e., Action 1, and the video output encoding FPS (φ), i.e., Action 2.

Conversely, $\boldsymbol{a}_2$ includes the CPU allocated to vRAN (c_r), i.e,. Action 4, the MCS value (ω) defined before as Action 5, and the bandwidth allocated to each UE, $\boldsymbol{\rho} = \{\rho_1, \rho_2, \cdots \rho_M\}$ as defined in Action 6, where M is the maximum number of users supported in the system. Here, the CPU and the radio (RB) resources are capacity constrained, i.e., $c_v + c_r \leq B_c$ and $\rho_1 + \rho_2 + \cdots \rho_M \leq B_\rho$, where B_c and B_ρ are, respectively, the total available CPU and number of RBs that can be allocated. To avoid clutter, we replace c_v and c_r with a generic c_k that denotes the CPU allocated to service k, and let ρ_m be the number of RBs allocated to UE m. Mathematically,

$$\boldsymbol{a}_k = \begin{cases} (\beta, \varphi, c_k), & \text{if } k = 1, \\ (\omega, c_k, \boldsymbol{\rho}), & \text{if } k = 2\,. \end{cases} \tag{1}$$

Next, we discretize the quantity of capacity-constrained resources that can be allocated, and map each feasible

combination of action variables during the n-th monitoring slot into an action index $\boldsymbol{a}_1^{(n)} := \{1, 2, \ldots, N_\beta \cdot N_\varphi \cdot N_{c_v}\}$ and $\boldsymbol{a}_2^{(n)} := \{1, 2, \ldots, N_\omega \cdot N_{c_r} \cdot N_{\boldsymbol{\rho}}\}$, where N_i is the number of elements in the discretized version of action variable $i = \{\beta, \varphi, \omega, c_v, c_r, \boldsymbol{\rho}\}$. We remark that the action space is a mixture of variables that are inherently continuous (e.g., CPU, output bitrate, output framerate) and discrete (RBs, MCS). The primary reason for discretization of all action variables is the ease in framework implementation, since invoking different techniques for learning continuous and discrete variables will increase the system complexity by manifolds. Besides, such action definition limits the action space to a subset of discrete positive values with low cardinality, and it facilitates simultaneous selection of several resources with a single action. Instead, converting all action variables to take continuous values, and using a continuous variable-specific learning algorithm, would introduce rounding off errors in the final stage.

Reward. For a given service, KPI satisfaction is achieved when the allocated resources make the observed KPIs to meet their respective target values. However, beside meeting the target KPIs, it is essential to keep the observed KPIs as close as possible to the target KPIs; failing that, the system may perform better than required at the cost of extra resource consumption. Consequently, the choice of a reward function should be such that it equally accounts for all the KPIs for a given service and its value increases as the observed KPIs approach the corresponding thresholds and vice versa.

Let the observed values of the livecast KPIs, i.e., WVMAF and client buffer state, and of the vRAN KPIs, i.e., latency and packet loss rate for the m-th UE, be denoted by $\zeta_o^m, \sigma_o^m, \lambda_o^m, \mu_o^m$, while the corresponding target values be $\zeta_t, \sigma_t, \lambda_t, \mu_t$, respectively. We define the reward value for m-th UE, r^m, as the sum of the reward components pertaining to each service-specific KPI k in the n-th monitoring slot within the same decision window, as:

$$r^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}) = \begin{cases} r_\zeta^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}) + r_\sigma^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}), & \text{if } k = 1, \\ r_\lambda^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}) + r_\mu^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}), & \text{if } k = 2. \end{cases} \tag{2}$$

In the above expressions, $r_\zeta^m(\cdot), r_\sigma^m(\cdot)$ are the reward components from WVMAF and buffer state (resp.) for the m-th UE, given by:

$$r_{\text{KPI}}^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}) = \begin{cases} 1 - \text{erf}(\text{KPI}_o^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}) - \text{KPI}_t), \\ \qquad\qquad\qquad\qquad \text{if KPI is met} \\ \text{erf}(\text{KPI}_o^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}) - \text{KPI}_t), \qquad \text{else}. \end{cases} \tag{3}$$

The terms $r_\lambda^m(\cdot)$ and $r_\mu^m(\cdot)$ are instead the reward components from latency and packet loss rate (resp.), which are given by similar expressions but with $(\text{KPI}_t - \text{KPI}_o^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}))$ as an argument of the erf function, since all values of latency and packet loss rate lower than their respective target values are acceptable. Since the minimum and maximum values of the erf function lie between −1 and +1, we have: $-2 \leq r^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(n)}) \leq 2$. For the individual reward components, in the positive region of operation, i.e., when the KPI threshold is met, the reward value is positive and it further increases to its maximum value +1 as the observed KPI approaches its target KPI value. Likewise, in the negative region of operation, i.e., when the KPI threshold is not met, the value of the individual reward components is negative, which further reduces and saturates to the minimum value −1 as the observed KPI moves away from the KPI threshold.

We recall that while devising the greedy resource allocation policy, the goal of the RL agent is to maximize the cumulative reward measured as the sum of immediate reward and future rewards over a long time horizon. To this end, we consider a generic decision window h and, extending the previous notation, we let $a_k^{(h-1)}$ denote the action for the k-th service selected in decision window $(h-1)$ and applied in decision window h. We then define the average reward over h, considering all the UEs, as

$$\bar{r}(\boldsymbol{y}^{(h)}, \boldsymbol{a}_k^{(h-1)}) := \frac{1}{MN} \sum_{n=1}^{N} \sum_{m=1}^{M} r^m(\boldsymbol{y}^{(n)}, \boldsymbol{a}_k^{(h-1)}), \tag{4}$$

where $\boldsymbol{y}^{(h)}$ is the vector of shared contexts observed in the N monitoring slots in decision window h, while $\boldsymbol{a}_k^{(h-1)}$ is the action for service k selected in decision window $h-1$ and applied in decision window h. Finally, we adopt the definition of cumulative reward for the k-th service, observed during decision window h, as the differential return $\boldsymbol{G}_k^{(h)}$ defined in [31] (see Appendix A in the Supplemental Material).

Estimation of user buffer states. A key challenge, however, is to estimate the actual buffer dynamics *without* explicit feedback from the users. Thus it is important to design an effective *online learning* mechanism. To this end, we keep track of the time status information provided by the video encoder and the sequence number of TCP acknowledgments, all information locally available. By monitoring the amount of bytes successfully delivered, the corresponding timestamp of the scenes been transmitted, and the encoded video's frame rate, we can estimate the buffer dynamics at the client's side using simple queuing theory. Fig. 8 shows the evolution over time of a video player's buffer state (ground truth) vs. the inferred value for 3 trivially chosen videos and system configuration parameters. In most of the cases, our inference method is remarkably accurate. This is the case for the first two subplots in Fig. 8. However, we have found that there are some small number of cases where there is a non-negligible inference error. We show an example of this in the right-most plot of the figure, where we have an error of almost one second. Fortunately, these always occurs for non-critical cases, i.e., cases where the buffer state never approaches zero (the state we want to avoid). Since our estimator is pessimistic, the outcome are simply more conservative decisions. We hence conclude that our inference approach is valid to compute reward.

Action-value estimation and action selection. At the end of the generic decision window h, actions need to be evaluated and the best one has to be selected. To this end, we compute the mean shared context over the N monitoring slots in h as

$$\bar{\boldsymbol{y}}^{(h)} = \sum_{n=1}^{N} z_n \boldsymbol{y}^{(n)} / \sum_{n=1}^{N} z_n, \tag{5}$$

where $z_n > 0$ and $z_N > z_{N-1} > \cdots > z_1$ are the weights assigned so that the latest shared context has the highest

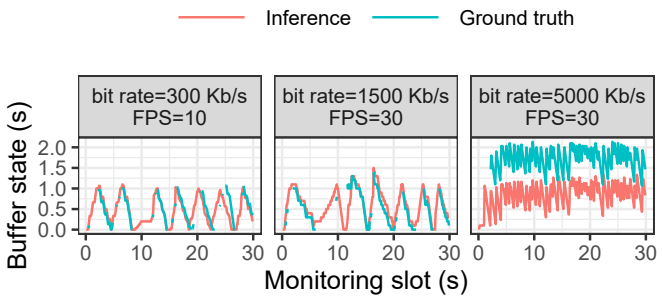


Fig. 8. Livecast client's buffer state inference

weight.[4] We then quantify the goodness of taking an action in response to the mean shared context using action values. For service k, the value of $\boldsymbol{a}_k^{(h)}$ given policy π_k, which is $q_{\pi_k}(\overline{\boldsymbol{y}}^{(h)}, \boldsymbol{a}_k^{(h)})$ (see Appendix A in the Supplemental Material), is defined as the expected differential return conditioned on $\overline{\boldsymbol{y}}^{(h)}$ and $\boldsymbol{a}_k^{(h)}$, following policy π_k, i.e.,

$$q_{\pi_k}(\boldsymbol{y}, a) = \mathbb{E}_{\pi_k}[\boldsymbol{G}_k^{(h)} | \overline{\boldsymbol{y}}^{(h)} = \boldsymbol{y}, \boldsymbol{a}_k^{(h)} = a]. \quad (6)$$

Since the context space $\mathcal{X}$ falls in the domain of real numbers, we use a practical method for action-value estimation using function approximation in an F-dimensional space, yielding the approximated function $\hat{q}_{\pi_k}(\overline{\boldsymbol{y}}^{(h)}, \boldsymbol{a}_k^{(h)}, w) = \sum_{f=1}^{F} w_f s_f(\overline{\boldsymbol{y}}^{(h)}, \boldsymbol{a}_k^{(h)})$, where $\mathbf{w}$ and $\mathbf{s}(\overline{\boldsymbol{y}}^{(h)}, \boldsymbol{a}_k^{(h)})$ denote the F-size weight and feature vectors (resp.), with the latter being generated using tile coding [32] (see Appendix B in the Supplemental Material).

The estimation of the action values is followed by an ϵ-greedy action selection policy [31], which selects the best action for each service so as to maximize its cumulative reward over an infinite time horizon. We consider an ϵ-greedy action selection with $\epsilon = 0.5$ and ϵ-decay factor = 0.999. The ϵ parameter decays by a factor of 0.999 in the subsequent decision period. This favors higher exploration while the environment is still unfamiliar; with progression of time, instead, it allows for further exploitation of the environment knowledge gained during the exploration, so as to maximize the expected return.

Discussion. While single-agent RL approaches can easily solve the above capacity constraints, they suffer from the curse of dimensionality, even if implemented with deep neural networks (see DQNs) [33]. As a result, the key challenge is to provide safe (i.e., within a set of hard constraints) and fair resource allocation decisions with a distributed multi-agent RL model that is amenable to scalable orchestration.

In this way, our distributed approach allows us to handle two sets of capacity constraints:

1) Computing resources: VERA can handle multiple vBSs (or multiple radio slices within a vBS), and multiple edge services that are competing for the same computing resource budget;
2) Radio resources: VERA can handle multiple users sharing a common carrier bandwidth.

4. Although they can be arbitrarily set, we fix them to $1, \ldots, N$, in accordance with the temporal sequence of the monitoring slots.

Algorithm 1: Fair Pareto-efficient Resource allocation

$S = \{\tilde{a}_k\}_k$,
$\{c_k, \rho_m(k)\ \forall m \in \mathcal{M}\} \leftarrow \tilde{a}_k(S), S' = \{c_k, \rho_m(k)\ \forall m \in \mathcal{M}\}, \forall k \in C$
/* Extract capacity-constrained actions from greedy action set $\{\tilde{a}_k\}$ */
if $\sum_{k \in C} c_k \le B_c$ and $\sum_{m \in \mathcal{M}} \rho_m(k) \le B_\rho, \forall k \in C$
/* Capacity-constraint check on the primary and secondary resource */
then
  $S^\star = S'$
  /* Output: Fair Pareto-efficient solution */
else if $\sum_{k \in C} c_k \le B_c$ and $\sum_{m \in \mathcal{M}} \rho_m(k) > B_\rho$, for any $k \in C$
/* Primary resource budget constraint met, secondary resource budget constraint not met */
then
  $\{\rho'_m(k)\ \forall m \in \mathcal{M}\} \leftarrow$ ParetoBlock($\{\rho_m(k)\ \forall m \in \mathcal{M}\}$), for the considered $k \in C$
  /* Revised Pareto-efficient fair secondary resource allocation adhered to budget constraint and allocated CPU */
  $S'' = \{c_k, \rho'_k(m)\ \forall m \in \mathcal{M}\}, \forall k \in C$
  $S^\star = S''$
  /* Output: Fair Pareto-efficient solution */
else if $\sum_{k \in C} c_k > B_c$
/* Primary resource budget constraint not met */
then
  $\{c'_k\} \leftarrow$ ParetoBlock($\{c_k\}$), $\forall k \in C$
  /* Revised Pareto-efficient fair primary resource allocation adhered to its budget constraint */
  $\{\rho'_m(k)\ \forall m \in \mathcal{M}\} \leftarrow$ ParetoBlock($\{\rho_m(k)\ \forall m \in \mathcal{M}\}$), $\forall k \in C$
  /* Revised Pareto-efficient fair secondary resource allocation adhering to revised Pareto efficient fair primary resource allocation */
  $S'' = \{c'_k, \rho'_m(k)\ \forall m \in \mathcal{M}\}, \forall k \in C$
  $S^\star = S''$
  /* Output: Fair Pareto-efficient solution */

Algorithm 2: ParetoBlock

Input: $\{c_k\}\ \forall k \in C$ s.t. $\sum_{k \in C} c_k > B_c$ or $\{\rho_m(k)\ \forall m \in \mathcal{M}\}$ s.t. $\sum_{m \in \mathcal{M}} \rho_m(k) > B_\rho\}$
/* Primary or secondary resource allocation violating the budget constraints */
$S_1 = \{c_k\}\ \forall k \in C$ or $S_1 = \{\rho_m(k)\}\ \forall m \in \mathcal{M}$, as applicable
$\mathcal{S}_e = \{S_1, S_2, \ldots\}$
/* Build expanded solution set */
$\mathcal{S}_s \leftarrow \{S_i/|C|\}_{\mathcal{S}_e}$ or $\mathcal{S}_s \leftarrow \{S_i/|\mathcal{M}|\}_{\mathcal{S}_e}$, as applicable
/* Rescale expanded solution set */
for $S \in \mathcal{S}_s$ **do**
  $\widehat{\mathcal{S}}_s \leftarrow \{\tilde{a}_k(S)\}$
/* Define refined actions set wrt $\mathcal{S}_s$ */
Create $\mathcal{S}_d$
/* Pareto dominant solution set */
Choose S'_1
/* Fair Pareto-efficient resource allocation */
Return: $S'_1 = \{c'_k\}\ \forall k \in C$ or $S'_1 = \{\rho'_m(k)\}\ \forall m \in \mathcal{M}$, as applicable

Although for the sake of clarity the above text and Fig. 7 refer to two service learning agents only, one of which is a single vBS serving M UEs, the scalable multi-agent design of VERA allows for as many learning agents as services, vBSs, or slices under the above capacity constraints.

4.3 Pareto analysis

We recall that the CPU and RB allocation for (resp.) service k and UE m are capacity-constrained resources. Hence, it is essential that the sum of CPU (RB) allocated to different services (UEs) does not exceed the available resource budget and that the selected actions can be enacted. To this end, we introduce an algorithm that works on the multi-dimensional actions selected by the ϵ-greedy policy in the RL framework introduced above, and it further refines them so that the resulting actions (*i*) meet the budget constraint and (*ii*) entail fair Pareto-efficient resource sharing.

It is important to note here that not only the CPU allocation across the services and RB allocation across the UEs are capacity constrained, the RB allocation is also dependent on the CPU allocated to vRAN. Thus, CPU and RB (resp.) act as the primary and secondary capacity constrained resources for vRAN. Unlike vRAN, the livecast service has no associated secondary capacity constrained resource. However, for the sake of mathematical proofs, this observation can be generalized as follows: the primary capacity constrained resource (here CPU) is distributed among K services, and each service may in turn serve M units (UEs for vRAN, none for livecast) using the primary resource. The secondary capacity constrained resource is distributed among M units of the service (if applicable). Further, the QoS satisfaction of each service in entirety comprises QoS satisfaction of individual units, and depends both on primary and secondary capacity constrained resource allocation.

To model such interdependence, we formulate the fair Pareto-efficient allocation of CPU across the services and RBs across the UEs in a given decision window as a constrained joint multi-criteria optimization problem. Further, for notational simplicity, we assume that each decision window comprises of just one monitoring slot, i.e., $N = 1$. Let $C, \mathcal{M}$ denote the set of services and UEs; given a set of coefficients $u_k \geq 0, k \in C, m \in \mathcal{M}$, with $\sum_{k\in C} u_k = 1$, it is required to find a solution $S^\star = \{c_k^\star, \rho_m^\star(k) \ \forall m \in \mathcal{M}\}, \forall k \in C$, that maximizes $\sum_{k\in C} u_k \mathbf{\Gamma}_k(S)$ such that $S \in \mathcal{S}_c$, $\sum_{k\in C} c_k \leq B_c$ and $\sum_{m\in\mathcal{M}} \rho_m(k) \leq B_\rho$. Here, c_k is the primary capacity constrained resource allocated to k-th service, $\rho_m(k)$ denotes the secondary capacity constrained resource allocated to m-th unit of the k-th service, $\mathcal{S}_c$ is the set of feasible capacity constrained resource allocations and $\mathbf{\Gamma}_k(S)$ is the criteria function denoting the reward of the k-th service in a decision period following the CPU allocation strategy S.

The flow of fair Pareto-efficient resource allocation is summarized in Alg. 1. The key component of Alg. 1 is ParetoBlock (Alg. 2) that solves the joint multi-criteria optimization problem. It is invoked whenever the sum of allocated primary (secondary) capacity constrained resource exceeds its specified budget. It initially considers the CPU (RB) allocation to the services (UEs) provided by the greedy resource allocation policy, and creates the expanded CPU (RB) allocation solution set by considering all possible values for the c_k's ($\rho_m(k)$'s) that are greater than those output by the greedy policy and whose sum does not exceed $|C|$ ($|\mathcal{M}|$) times the available budget. Such values are then scaled by $|C|$ ($|\mathcal{M}|$), to get candidate allocation values that meet the CPU (RB) budget. The corresponding action set, $\widehat{\mathcal{S}}_s$, is built starting from such c_k's ($\rho_m(k)$'s) and possibly refining the actions so that their components take feasible values considering the dependence of primary and secondary resource. Such actions, $\{\widehat{\boldsymbol{a}}_k(S)\}$, $\mathcal{S} \in \widehat{\mathcal{S}}_s$, are then used to compute the values of $\mathbf{\Gamma}_k(S)$ to identify the Pareto-dominant solution set through iterative search and update, $\mathcal{S}_d \leftarrow \{S\}$, s.t. $\forall S \in \mathcal{S}_d$, $\forall S' \in \widehat{\mathcal{S}}_s, \mathbf{\Gamma}_i(S) > \mathbf{\Gamma}_i(S'), \mathbf{\Gamma}_j(S) \geq \mathbf{\Gamma}_j(S'), \forall i, j \in C(\mathcal{M}), i \neq j$. Finally, for the primary capacity constrained resource, the Pareto-dominant solution that maximizes the minimum value of criterion function, i.e., the reward value over all the services is chosen as the fair Pareto-efficient solution. For the secondary capacity constrained resource, we define $g(m) := l_t(m) - l_i(m) \ \forall m \in \mathcal{M}$, where $l_t(m), l_i(m)$ denote (resp.) the target traffic load and the instantaneous rate achieved by m-th UE. Further, we choose the secondary resource allocation $\{\rho'_m(k) \ \forall m \in \mathcal{M}\}$ for a given fair primary resource c'_k as the solution that minimizes $\max_{m\in\mathcal{M}} g(m)$ over all $S \in \mathcal{S}_d$.

We finally prove the following results:

- The solution $S^\star$ introduced above is Pareto-efficient with respect to the primary (see Proposition 1), as well as jointly Pareto-efficient with respect to primary as well as secondary capacity constrained resources (see Proposition 2);
- Alg. 2 converges to a Pareto-efficient solution set, at a sub-linear rate (see Proposition 3);
- Alg. 2 converges to a solution that is fair with respect to the primary capacity constrained resource (see Proposition 4), as well as the secondary capacity constrained resource for a given primary capacity constrained resource allocation (see Proposition 5), thus leading to a fair Pareto-efficient solution.

Proposition 1. *Pareto-efficient allocation of the primary resource: Given a set of coefficients $u_k \geq 0, k \in C$, such that, $\sum_{k\in C} u_k = 1$, then the solution $S^\star = \{c_k^\star\}$, $k \in C$, that maximizes the multi-criteria optimization problem $\sum_{k\in C} u_k \mathbf{\Gamma}_k(S)$, is Pareto-efficient.*

Proof. See Appendix C in the Supplemental Material. □

Proposition 2. *Pareto-efficient joint allocation of primary and secondary resource: Given a set of coefficients $u_k \geq 0, k \in C$, such that, $\sum_{k\in C} u_k = 1$, then the solution $S^\star = \{c_k^\star, \rho_m^\star(k) \ \forall m \in \mathcal{M}\}, \forall k \in C$, that maximizes the multi-criteria optimization problem $\sum_{k\in C} u_k \mathbf{\Gamma}_k(S)$, is Pareto-efficient.*

Proof. See Appendix C. □

Proposition 3. *Alg. 2 converges to a Pareto-efficient solution set at a sub-linear rate.*

Proof. See Appendix C in the Supplemental Material. □

Proposition 4. *Fairness of Pareto-efficient primary resource allocation: The solution $S^\star = \{c_k^\star, \rho_m^\star(k) \ \forall m \in \mathcal{M}\}, \forall k \in C$ obtained using Algorithm 2 is fair with respect to primary resource allocation $c_k^\star, \forall k \in C$.*

Proof. See Appendix C in the Supplemental Material. □

Proposition 5. *Fairness of Pareto-efficient secondary resource allocation in the vRAN: For a given fair primary resource allocation $c_k^\star$ in the solution $S^\star = \{c_k^\star, \rho_m^\star(k) \ \forall m \in \mathcal{M}\}$, for $k = 2$ (denoting the vRAN service), $S^\star$ is fair with respect to secondary resource allocation $\{\rho_k^\star(m) \ \forall m \in \mathcal{M}\}$.*

Proof. See Appendix C in the Supplemental Material. □

We remark that two-stage solutions for resource allocation via distributed RL have often been applied in the literature, where greedy global solutions are obtained in the first stage, and then each agent tunes its own parameters in the second stage to ensure fairness. However, compared to such existing schemes, our proposed VERA framework has several unique features: (i) it jointly considers capacity-constrained resources and service specific operating parameters, (ii) it accounts for the interdependence of primary and

secondary capacity-constrained resources, (iii) it envisions a novel Pareto block design for ensuring Pareto-optimal fair action selection for virtual services hosted at the network edge. To the best of our knowledge, none of the existing works have addressed these issues.

4.4 Learning algorithm

We exploit the concept of experience-based learning using sample sequences of shared context, actions, and rewards observed from the actual interaction of the RL agent with the environment. SARSA, an acronym for quintuple $(S_t, A_t, R_t, S_{t+1}, A_{t+1})$, is an on-line policy algorithm where learning of the RL agent at time t is governed by its current state S_t, choice of action A_t, reward R_t received on taking action A_t, state S_{t+1} that the RL agent enters after taking action A_t, and finally the next action A_{t+1} that the agent chooses in new state S_{t+1} [31]. We here highlight that compared to Q-learning, another commonly used learning algorithm, SARSA adopts a conservative learning approach by avoiding high-risk actions that may generate large negative rewards from the environment. This is especially relevant since the applicability of VERA framework may be extended to ultra low latency and ultra reliable services wherein ensuring radio connectivity is critical and any violation of KPI targets would incur high costs. Although double Q-learning, a Q-learning variant, is also conservative, it is computationally more intensive than SARSA owing to the requirement of computing and updating two Q-policies simultaneously. Further, SARSA has low per-sample variance, which makes it less susceptible to convergence problems [31]; hence, future extensions of the VERA framework can be easily upgraded to deep networks if necessary.

For clarity and without loss of generality, here we focus on the learning of a single RL agent that corresponds to one of the services, over successive decision windows. Given the mean shared context and possible actions, the primary steps in the learning algorithm are: (i) obtain greedy resource allocation policy for service k through estimation of action values $q_{\pi_k}(y, a)$, (ii) obtain a fair Pareto-efficient resource allocation policy by collating and returning the greedy policies of all the services, and (iii) update of the action-value estimates for service k using differential semi-gradient SARSA [31].

4.5 Computational complexity analysis

We now discuss the complexity analysis of the VERA framework implementation. The most complex operations in the whole framework are given by the following steps: (i) greedy action selection for the K services, (ii) joint Pareto-efficient fair allocation of primary resource among K services, and secondary resources among M units of each service, (iii) computation of weighted mean of the context and mean reward for K services in a decision window comprising N monitoring slots, and (iv) update of the weight vector for learning radio policy based on the KPIs observation from the K services. Corresponding to each of these steps, the computational complexities of taking the resource allocation decision once in the VERA framework are given by $O(|\mathcal{A}|)$ (with $|\mathcal{A}|$ being the cardinality of the VERA action space), $O(KM)$, $O(KNM)$, and $O(K)$, respectively. Hence, the overall complexity is $O(|\mathcal{A}|)+O(KM)+O(KNM)+O(K) \approx O(|\mathcal{A}|)$ since the first term is the dominant one as $|\mathcal{A}|$ is much larger than KNM. Thus, computations in VERA scale linearly with the increase in the cardinality of the action space.

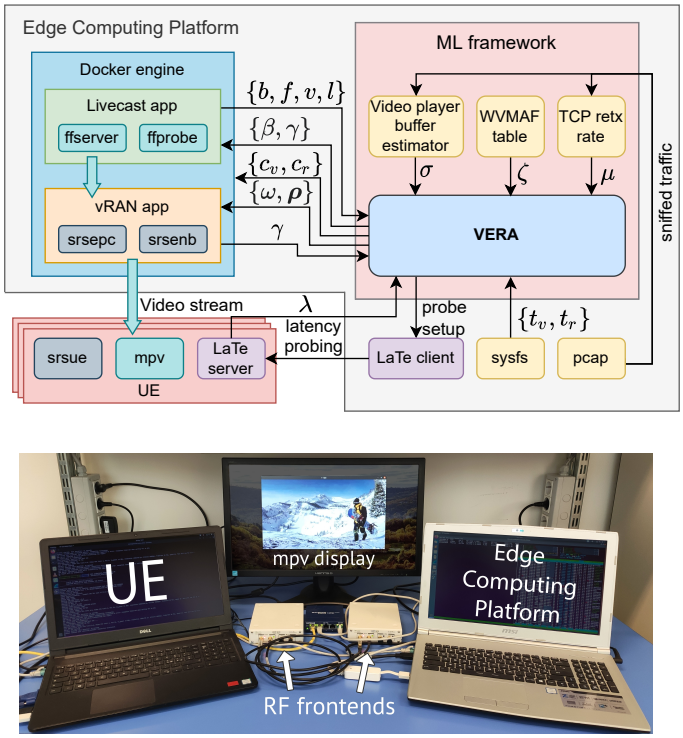


Fig. 9. (top) VERA system representation (details in Sec. 5.1); (bottom) picture of the testbed (details in Sec. 5.2)

5 Proof-of-concept Implementation

In this section, we first introduce our proof-of-concept implementation (Sec. 5.1), and then we present the parameters and settings we use to collect our datasets and run our experimental tests (Sec. 5.2).

5.1 VERA implementation

As depicted in Fig. 9, we have integrated VERA in a testbed based on `srsRAN` (to emulate a vRAN service), `ffserver` (to emulate a livecast video service), and `mpv` (livecast video client deployed at each UE video player).

VERA's learning agents receive real-time context information directly from the vRAN, the livecast service, and the edge computing platform, using custom-made TCP-based interfaces. More specifically, the CQI information is retrieved from srseNB using its `enb_metrics_interface` class. The input video features (i.e., input video FPS, bitrate, and resolution) are instead probed using `ffprobe`, which is part of `ffmpeg`, and then sent to VERA whenever a new video source is selected. The CPU throttling times of the livecast and vRAN services are collected by interfacing with Linux `cgroups`, using Linux' pseudo file-system `sysfs`. Finally, the network load, which corresponds to the offered load that the vBS has to process, is derived by summing up

TABLE 2
Input and output video characteristics

Video characteristics	Input	Output
Resolution [pixels]	1920×1080	1280×720
Bit rate [Mbps]	18	0.3, 1.5, 5, 10
Frame rate [FPS]	30	10, 20, 30
Codec (container)	VP9 – WebM	VP9 – WebM

the bytes produced each second by the video encoder, as reported in its output log.

Concerning the reward signal, the observed unidirectional latency, the packet loss rate, the estimated video player buffer state, and the WVMAF are sent to VERA in real time, using additional custom TCP-based interfaces. In more detail, the latency measurements are obtained using LaTe[5], a flexible client-server multi-protocol Latency Tester that sends probes to the network under test and measures the delay that the probe experiences. To this end, clock synchronization between the Edge Platform and the UEs is performed using Precision Time Protocol daemon (PTPd).

The packet loss rate, computed through the TCP segment retransmission rate, and the buffer occupancy are estimated at the edge platform by leveraging, respectively, the TCP sequence and the acknowledgment numbers obtained using `libpcap`. To model the player buffer and, hence, infer its status, VERA exploits the TCP acknowledgment numbers, the offered load, and the output frame numbers of the video encoder, as explained in Sec. 4. Finally, to compute the WVMAF, we use a lookup table that maps every choice of encoding parameters to the expected VMAF score. Calculating the VMAF score is indeed a computing-intensive task that cannot be performed in real-time without a noticeable performance impact. The table has been built by considering a collection of 1080p video samples[6], and by encoding each source video using every encoding parameters combination available to VERA. The VMAF score is calculated by comparing the frames of each encoded video to the original frames. The WVMAF score of each video sample is obtained by multiplying its VMAF score by the ratio of the output frame rate and the input frame rate. Then, for every combination of the encoding parameters, the corresponding WVMAF score is computed averaging the WVMAF scores of all video samples encoded with the same combination of parameters.

At last, to enforce decisions made by VERA, the per-UE RB allocation and the MCS policy are sent to the vRAN through a custom interface, the CPU allocation is set through Docker API (which, in turn, enforces it using Linux cgroups), and the video encoding FPS and bit rate policies are updated overriding the parameter settings in the livecast service, thus allowing VERA to work in real time.

5.2 Testbed configuration

The testbed configuration used to collect the necessary dataset and run the VERA framework in real time is based on the architecture presented in Sec. 2.

5. https://github.com/francescoraves483/LaMP_LaTe
6. https://media.xiph.org/video/derf/

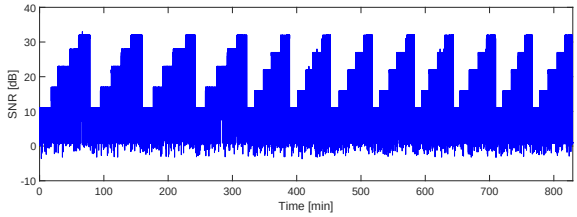


Fig. 10. Time evolution of SNR in our experiments

The edge computing platform and the UEs are hosted on GNU/Linux machines; they accommodate, respectively, an Intel i7-7700HQ and an Intel i7-8550U CPU, with 16 GB of DDR4 memory. The LTE network uses a 10-MHz channel in band 7, which provides a capacity of 50 RBs. The dynamic SNR pattern, considered in our experiments to be experienced by the UEs, is depicted in Fig. 10; the values of SNR are then mapped into CQI by the vRAN system. We assume a network slice dedicated to the livecast service, with a capacity that may vary between 12 and 36 RBs. As RF frontend, Ettus USRP B210 boards are used to perform up/down-conversion, filtering, amplification, and AD/DA conversion of the UEs and eNB LTE signals.

As mentioned above, we use `ffmpeg`, as this is compatible with a wide set of video codecs, picture formats, containers, besides offering a number of filters to modify video characteristics. Tab. 2 reports the characteristics of the input and output videos in our experiments. The characteristics of the input videos are representative of a livecast content, as they ensure that the video can be properly played by the client player in a wide range of network conditions and client configurations. The output parameters have been set so as to allow for the best video quality retention, hence a good level of user Quality of Experience, while requiring a reasonable consumption of network and computing resources. Tab. 3 includes additional parameters settings that we have used at the video encoder, server, and client, which are typical of a livecast service.

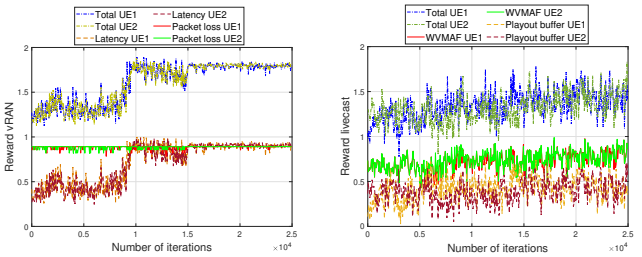
Finally, we consider that decisions are made every monitoring slot ($N = 1$), and, unless otherwise specified, we set the available CPU budget to 3 vCPUs. We would like to highlight that the computation cost of our solution can be fully sustained by our small-size testbed. On average, one iteration (i.e., metrics parsing, action selection, reward computation and weights update) requires 16.1 ms. Over one thousand iterations, we measured a maximum iteration time of 25.8 ms, with a 99-th percentile below 18.6 ms. This corresponds to an average CPU load below 5% when the monitoring slot is 100-ms long.

6 Evaluation and Experimental Validation

In this section, we first present the numerical results (Sec. 6.1) derived using the datasets obtained through extensive experiments on the testbed described in Sec. 5; and then, we present the performance of a real-time implementation of VERA on the testbed (Sec. 6.2). Here, we would like to highlight that since the existing datasets do not consider the context variables and action space of our interest, they fail to exhibit the non-trivial correlations that we observed in our experiments. Thus, for the following results we rely on our self-collected datasets.

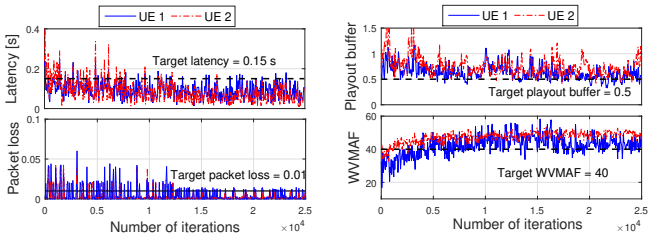
TABLE 3
Video encoder, server and client parameters

Encoder/server param.	Description	Value
StartSendOnKey	If set, the video is streamed starting from the first I-frame generated by the encoder, i.e., P-frames not preceded by an I-frame are discarded	Enabled
Preroll N	The video is streamed starting not from the most recent frame but from N seconds in the past; if set, it increases buffer occupancy at the expense of the end-to-end latency	Disabled
VP9 Threads	No. of threads that decoder & encoder can use: high values increase speed if multiple vCPUs are allocated, at the cost of a small overhead.	No. of allocated vCPUs
VP9 Quality	Possible settings: realtime, good, or best. It controls the time that the encoder can take to encode frames beyond their presentation time	Realtime (no additional time beyond presentation timestamp)
VP9 Speed	It controls the trade-off between computational lightness and picture quality. Possible values in [0,16], with the higher values prioritizing encoding speed (i.e., lower CPU consumption) over picture quality	16
Client CachePauseInitial	The client pauses the playback at the beginning to wait for the buffer to fill, so as to avoid pauses while the video is playing	Enabled
Client CachePauseWait	Video time that the client requires before resuming the playback when paused. It affects the end-to-end latency, but a bigger size can better cope with oscillations in the data transfer and encoding delays	1 s



(a) (b)

Fig. 11. Convergence of reward values: vRAN (a) and livecast (b) services



(a) (b)

Fig. 12. KPI evolution with respect to iterations: (a) vRAN and (b) livecast. Dashed dark line shows target KPI values.

6.1 Numerical results

The baseline scenario we consider in our numerical performance evaluation includes 1 vBS, 2 UEs, and a livecast service streaming a single video to both UEs. The CPU and RB budgets are fixed to 2 vCPUs and 60 RBs, respectively.

Convergence evaluation. Fig. 11 depicts the time evolution of reward values for the vRAN and livecast services in the baseline scenario. From the plots, we observe that, despite the large heterogeneous action set and the diverse context vector, the reward corresponding to each of the KPIs, and hence the total reward, saturates close to the maximum value for both the UEs, thereby highlighting the efficient learning capability of the VERA framework. Also, the convergence of the livecast service is relatively slower with respect to vRAN owing to its slowly varying dynamics.

KPI performance. Next, Fig. 12 presents the evolution of the KPIs across iterations during the learning process. Notice that the KPI satisfaction for vRAN is achieved when its latency and packet loss do not exceed their respective targets. On the contrary, for the livecast service, the playout buffer and WVMAF should not fall below their target values, while keeping the KPIs observed for both the services as close as possible to their target values. According to the 3GPP 5G specifications [30] and acceptable QoE, the target KPI values are set at 150 ms, 0.01, 0.5 s and 40 (resp.) for latency, packet loss, playout buffer, and WVMAF. From the plots, we observe that barring a few initial iterations during which the algorithm is still learning, the choice of actions by the VERA framework leads to KPI satisfaction for both vRAN and livecast services. To quantify VERA's suboptimality, the mean KPI target violation for VERA post convergence of the algorithm is 3.7%.

Performance under different constraints. We now evaluate the impact that different CPU and RB capacity constraints have on the performance of VERA. To this end, we consider two additional scenarios having budgets 3 vCPUs - 60 RBs and 2 vCPUs - 54 RBs, along with the baseline scenario. The performance is characterized using a distance parameter $d_{KPI} = \text{KPI}_o - \text{KPI}_t$, where KPI_o is the KPI target threshold and KPI_t is the KPI experienced at time t, and which basically quantifies how far away the observed KPI value is from its target. Fig. 13 compares the average value of d_{KPI} computed over both UEs for the said scenarios, for both vRAN and livecast services. It may be noted that a negative (positive) value of d_{KPI} for vRAN (livecast) denotes KPI satisfaction, and, irrespective of the service type, it is desirable that d_{KPI} is as close as possible to 0. From the plots, we observe that when the budget constraints are stringent, the choice of actions in order to meet the KPI target values is limited. Consequently, the resource

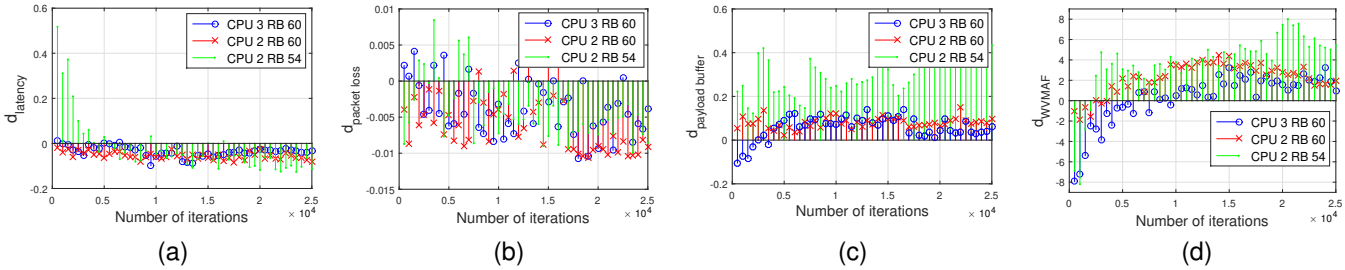


Fig. 13. Performance of VERA under varying CPU and RB budget constraints: (a) latency, (b) packet loss, (c) playout buffer, and (d) WVMAF

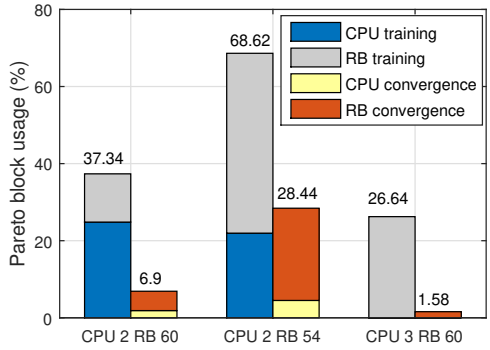


Fig. 14. Utilization of Pareto block in VERA during training and after convergence has been attained

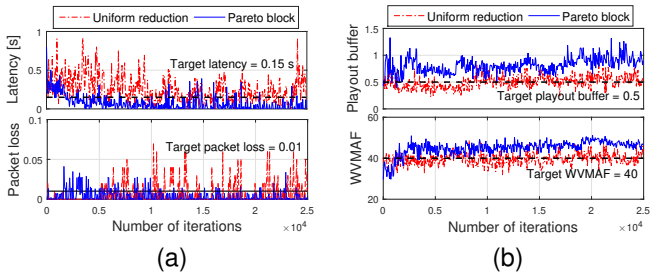


Fig. 15. KPI comparison of VERA with Pareto block and uniform reduction approach: (a) vRAN, (b) livecast

allocation efficiency is slightly compromised as shown by higher d_{KPI} values. Nevertheless, the KPI satisfaction is still achieved. As more resources are made available in terms of CPU and RBs, d_{KPI} values cling closer to 0, thereby minimizing the wastage in resource allocation. Thus, VERA can successfully attain KPI satisfaction for both the services and UEs under varying CPU and RB capacity constraints, however, stringent constraints may lead to a marginal loss in efficiency.

Pareto block statistics. Next, we investigate the significance of the Pareto block in the VERA framework. The bar plot in Fig. 14 shows the statistics of the Pareto block usage under different CPU and RB capacity constraints. We observe that the Pareto block usage for CPU as well as RB is the highest when budget is the most stringent, i.e., 2 vCPUs - 54 RBs. However, on a positive note, the Pareto block invocation substantially reduces once VERA has attained convergence compared to its training phase. This in turn suggests that, when a pre-trained VERA model is used, the Pareto block will not add to the system runtime complexity.

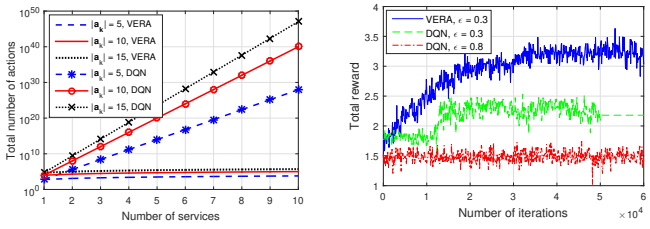


Fig. 16. Performance of VERA in comparison to DQN: (a) cardinality of action space, (b) total reward

To further emphasize this aspect, we replace the Pareto analysis in the VERA framework by a more intuitive and simpler uniform reduction approach, wherein an equal proportion of any excess CPU (RB) allocated beyond the budget is subtracted from the allocated CPU (RB) values across the services (UEs) such that the budget constraint is met. Fig. 15 presents the KPI (averaged over both UEs) comparison using the Pareto block and uniform reduction in the worst of our considered scenarios, i.e., 2 vCPUs - 54 RBs for vRAN and livecast services. From the plots we observe that unlike the Pareto block, uniform reduction fails to meet the target KPI values. This confirms that the Pareto block has a crucial role in optimal resource orchestration, especially when resources are constrained.

Comparison with other approaches. Finally, we address the scalability of the VERA framework. To emphasize the distributed decision making used by VERA, we compare its performance to a data-driven centralized framework using DQN. The choice of DQN for comparison against the VERA framework is motivated from the fact that an RL-based centralized solution (DQN in our case) for resource allocation at the network edge is a very widely used approach. Thus, it is the most relevant benchmark in the state-of-the-art. Since a generic DQN has no provision to enforce capacity constraints, to be fair, we consider a scenario wherein full CPU is available to the hosted services and there is no RB capacity constraint.

Before discussing the numerical results, we first characterize the scalability of VERA and DQN as follows. We generalize the notation of action vector for k-th service as $\boldsymbol{a}_k = \{a_{k1}, a_{k2}, \cdots, a_{kP}\}$, with the generic a_{kp} indicating the p-th action variable of the k-th service, which can assume values from a set of $N_{a_{kp}}$ discrete elements, with $P = |\boldsymbol{a}_k|$. Since DQN is a centralized framework, it handles the action vector corresponding to different services together, conse-

quently, $|\mathcal{A}|_{DQN} = \prod_{k=1}^{K} \prod_{p=1}^{|\boldsymbol{a}_k|} N_{a_{kp}}$. On the contrary, VERA handles different services in a distributed manner by assigning a separate learning agent for each service, leading to $|\mathcal{A}|_{VERA} = \sum_{k=1}^{K} \prod_{p=1}^{|\boldsymbol{a}_k|} N_{a_{kp}}$. For simplicity, assuming that all the services comprise the same number of action variables $|\boldsymbol{a}_k|$, and each action variable assumes a value from the set of discrete elements of the same cardinality $N_{a_{kp}}$, then $|\mathcal{A}|_{DQN} = N_{a_{kp}}^{|\boldsymbol{a}_k|^K}$, and $|\mathcal{A}|_{VERA} = K \times N_{a_{kp}}^{|\boldsymbol{a}_k|}$. Fig. 16a shows the variation of the total number of actions in VERA and DQN frameworks with increasing number of services, for $N_{a_{kp}} = 4$. It can be clearly observed that as the number of services hosted in the server increases, the cardinality of the action space for DQN rises very sharply in comparison to that of VERA, and this difference is even more evident as the number of action variables per service also increases.

Further, Fig. 16b shows the convergence of total reward from vRAN as well as livecast services observed in consequence to actions chosen by VERA and DQN. The choice of reward as a comparison metric helps to better characterize the scalability of the system. We observe that due to the high cardinality of the action space in DQN, it converges poorly. With the same ϵ-greedy action selection policy as VERA (i.e., $\epsilon = 0.3$, ϵ-decay = 0.9999), DQN is unable to explore all the actions while ϵ decays down to a negligibly small value and the learning agent gets caught up in a local optimum. Even if the action selection parameters are improved ($\epsilon = 0.8$, ϵ-decay = 0.9999), DQN still explores the action space until $\sim$ 60k iterations in our experiments, whereas VERA has shown a clean and early convergence with smaller ϵ value. To this end, it is worth noting that VERA exhibits a much higher scalability in terms of cardinality of action space compared to the state-of-the-art DQN-based centralized framework. To quantify the scalability performance, we define the scaling cost as the sum of reward deficit with respect to maximum reward value at convergence and the fraction of total iterations used for convergence. In our experiments, we found that the scaling cost of VERA is 45% and 60% lower compared to DQN, (resp.) for $\epsilon = 0.3$ and $\epsilon = 0.8$. We also compare the KPI performance of VERA with respect to DQN, which is included in the supplemental material due to space limitations here. To summarize, unlike the centralized DQN framework, VERA proposes an RL-based distributed learning scenario having service-specific learning agents. This limits the action space of each agent to the number of actions in the particular service. Consequently, the learning process is expedited and early convergence leads to better KPI performance, hence higher QoS satisfaction.

6.2 Proof-of-concept results

A pre-trained version of the VERA RL agents has been generated offline using the dataset collected from our testbed, then it has been evaluated online and in real time in two scenarios with different RBs availability, i.e., 24 and 36 RBs, and with CPU budget equal to 3 vCPUs. Notice that the use of pre-trained RL agents is only to expedite the learning curve at the inception of VERA framework in the testbed. In a real-world scenario, VERA uses differential semi-gradient SARSA for continuously learning from its environment over the time horizon. In both scenarios, a single UE connects to the vRAN and receives the livecast, using the configuration described in Sec. 5.2. For this case, the USRPs' transmission gain is set to a high value, so as to ensure that the SNR on uplink and downlink does not drop below 29 dB.

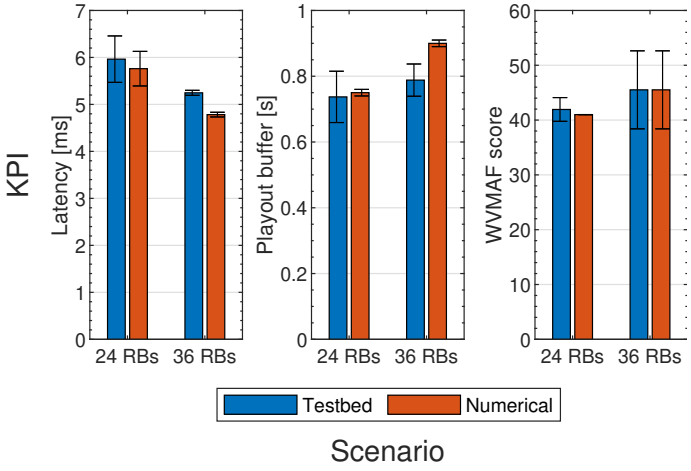


Fig. 17. Comparison of testbed and numerical KPI values for the livecast and vRAN services on the 24 RBs and 36 RBs scenarios with CPU budget equal to 3 vCPUs. Error bars indicate the confidence interval at 95% confidence level.

The radio latency, playout buffer and WMAF KPIs, collected from both testbed and numerical experiments, are compared in Fig. 17 (packet loss is omitted as it is equal to 0 in all cases). All KPIs are always satisfied for both the vRAN and the livecast service. As expected, the latency is higher in the case of 24 RBs than for 36 RBs, on the contrary, the playout buffer and WVMAF are less for 24 RBs than 36 RBs, owing to the lower number of radio resources being available in the first scenario: a higher number of allocated RBs leads to higher user throughput, which results in less latency, higher playout buffer and higher WVMAF score.

Importantly, the relative difference between testbed and numerical KPI values never exceeds 12.4%, with such a value being observed in the case of the playout buffer in the 36 RBs scenario. The similarity between testbed and numerical results validates VERA performance in a real time, hardware-in-the-loop implementation, and it demonstrates the effectiveness of our solution in a real-world environment.

7 Related Work

Several works have addressed the VNF placement problem at the network edge, which is related but orthogonal to the problem we face. Recent examples include: [34], which minimizes latency and system cost; [35], which optimizes both service placement and traffic routing under different resource constraints; and [36], which uses cooperation among edge nodes for service caching and workload scheduling.

Other studies have focused on QoE provisioning to mobile users through edge-assisted solutions. In particular, [37] presents an RL framework for crowdcasting services at the edge meeting bit rate as well as streaming and channel switching latency requirements, while minimizing the overall computing and bandwidth cost. [38], instead, designs and implements an edge network orchestrator, and a server

assignment and frame resolution selection algorithm for best latency-accuracy trade-off in mobile augmented reality.

Relevant to our work are also existing studies on resource consumption by edge user applications. In particular, [39] investigates the impact of real-time video analytics on computing and energy resources, while [40] focuses on image processing through CNNs and maximizes the learning accuracy given the limited resources at the edge.

The above literature does not consider the inherent resource contention between edge services and virtualized RANs. Related with this, [8] jointly optimizes the allocated resources to edge services and the placement of RAN functions, but uses an over-simplistic linear optimization model that cannot adapt to quick system dynamics. Like us, some other authors have used machine learning for practical resource allocation, radio parameter settings, and service KPI support in cellular networks. Among these, [41], [42], [43], [44], [45] aim at maximizing throughput through channel or link-rate selection, using multi-armed bandit techniques. Similarly, but leveraging the contextual information from the environment, [46] proposes an RL approach for rate selection and resource allocation, and [11] to maximize throughput subject to power consumption constraints. [10] extends the latter to accommodate service KPI constraints. However, [10] does not consider individual user dynamics nor fairness in resource allocation, as we do. RL-based schemes can also be found in [47], [48], [49], to minimize latency and packet drop rate in 5G systems. The work in [9], instead, tackles computing resource allocation in a virtualized radio access, and introduces a deep RL approach for resource management. Deep RL is also used to determine the suitable MCS and transmit power level in cognitive radio networks in [12] and [14], respectively, and to maximize the network sum-rate in [13]. Recently, [50] proposed a data-driven O-RAN-compliant framework that configures DUs/RUs according to a specified spectrum access policy.

More deep RL approaches are proposed for resource allocation in edge applications dealing with industrial IoT and internet of medical things [16], [17]. Other related resource allocation problems in computationally constrained scenarios such as joint server selection, task offloading and handover in multi-access edge computing wireless networks have been tackled through DQNs as in [18], [19], [20], [21]. Besides, solutions for other computation resource dependent problems like content caching [23], and network function placement in edge servers [22] have also been proposed using DQN. However, such works are subject to scalability issues in multi-service edge scenarios.

We underline that, unlike previous work, we address the allocation of edge resources *constrained to a limited budget across different, competing, virtual services*. Through our testbed, we identify the non-trivial correlations existing among the actions related to the different services, making the VERA learning objectives very different from those of existing works. To derive a scalable solution that can accommodate multiple services and vBSs (or slices of vBSs), we resort to a multi-agent RL mode. And, inspired by [24] and other literature on autonomous driving, we accommodate such hard constraints as a non-learnable building block. Different from previous work, however, we design a Pareto-efficient block for this task, which provides fair resource allocation across agents (vBSs and edge applications).

Finally, a preliminary version of our solution with only one user and considering only CPU as a constrained resource was presented in our conference paper [51].

8 Conclusions

We considered an edge computing platform hosting virtualized user applications and network services (namely, vRAN) competing for the same resources. We first investigated the correlations existing between the dynamics of such services through an experimental testbed that leverages a containerized livecast application and a containerized LTE base station. Then we developed a distributed learning framework, called VERA, that sets the configuration of both types of services so that the target KPIs can be met in spite of the limited availability of computing resources at the edge. Importantly, VERA also exploits a Pareto analysis that leads to fair Pareto-efficient decisions, and it can scale well with the number of virtualized services that are hosted at the edge platform. Our experimental results demonstrate the feasibility of the VERA approach and the important role of the Pareto analysis. Also, they show the excellent performance we can obtain in the presence of capacity-constrained resources with the KPI target violation limited to just 3.7%. Further, we show that VERA performs similarly when executed in our real-time proof-of-concept implementation, with KPI differences below 12.4%, thus confirming the effectiveness of VERA also in a real-world environment. We compare the performance of VERA to the centralized DQN framework and found it to be 54% more scalable, thereby establishing the efficacy of distributed over centralized learning in such complex resource limited scenarios. Finally, we are working on the integration of more user applications (e.g., robots control) with the livecast and vRAN services in the testbed, and further analysis in this respect is the scope of our future work.

References

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

[2] O. R. Alliance, "O-RAN: Towards an Open and Smart RAN," *White Paper*, 2018.

[3] Cisco, Rakuten, Altiostar, "Reimagining the End-to-End Mobile Network in the 5G Era," *White Paper*, 2019.

[4] Samsung, "Virtualized Radio Access Network: Architecture, Key technologies and Benefits." *Technical Report*, 2019.

[5] Intel, "vRAN: The Next Step in Network Transformation," *White Paper*, 2017.

[6] G. Garcia-Aviles *et al.*, "Nuberu: Reliable ran virtualization in shared platforms," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 749–761. [Online]. Available: https://doi.org/10.1145/3447993.3483266

[7] Nokia, "The edge cloud: An agile foundation to support advanced new services," *White Paper*, 2018.

[8] 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, 2018.

[9] J. A. Ayala-Romero, A. Garcia-Saavedra, M. Gramaglia, X. Costa-Perez, A. Banchs, and J. J. Alcaraz, "vrAIn: Deep learning based orchestration for computing and radio resources in vRANs," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020.

[10] J. A. Ayala-Romero *et al.*, "EdgeBOL: automating energy-savings for mobile edge AI," in *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, 2021, pp. 397–410.

[11] J. A. Ayala-Romero, A. Garcia-Saavedra, X. Costa-Perez, and G. Iosifidis, "Bayesian online learning for energy-aware resource orchestration in virtualized RANs," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[12] L. Zhang, J. Tan, Y. Liang, G. Feng, and D. Niyato, "Deep reinforcement learning-based modulation and coding scheme selection in cognitive heterogeneous networks," *IEEE Trans. Wireless Commun.*, vol. 18, no. 6, pp. 3281–3294, Apr. 2019.

[13] Y. S. Nasir and D. Guo, "Multi-agent deep reinforcement learning for dynamic power allocation in wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 10, pp. 2239–2250, Aug. 2019.

[14] X. Li, J. Fang, W. Cheng, H. Duan, Z. Chen, and H. Li, "Intelligent power control for spectrum sharing in cognitive radios: A deep reinforcement learning approach," *IEEE Access*, vol. 6, pp. 25 463–25 473, Apr. 2018.

[15] F. Wang, C. Zhang, F. wang, J. Liu, Y. Zhu, H. Pang, and L. Sun, "Intelligent edge-assisted crowdcast with deep reinforcement learning for personalized qoe," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 910–918.

[16] W. Cheng, X. Liu, X. Wang, and G. Nie, "Task offloading and resource allocation for industrial internet of things: A double-dueling deep q-network approach," *IEEE Access*, vol. 10, pp. 103 111–103 120, 2022.

[17] X. Yuan, Z. Zhang, C. Feng, Y. Cui, S. Garg, G. Kaddoum, and K. Yu, "A dqn-based frame aggregation and task offloading approach for edge-enabled iomt," *IEEE Trans. Netw. Sci. Eng.*, pp. 1–13, 2022.

[18] T. M. Ho and K.-K. Nguyen, "Deep q-learning for joint server selection, offloading, and handover in multi-access edge computing," in *ICC 2021 - IEEE International Conference on Communications*, 2021, pp. 1–6.

[19] F. Khoramnejad and M. Erol-Kantarci, "On joint offloading and resource allocation: A double deep q-network approach," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 4, pp. 1126–1141, 2021.

[20] Y. Li, F. Qi, Z. Wang, X. Yu, and S. Shao, "Distributed edge computing offloading algorithm based on deep reinforcement learning," *IEEE Access*, vol. 8, pp. 85 204–85 215, 2020.

[21] Y.-C. Wu, T. Q. Dinh, Y. Fu, C. Lin, and T. Q. S. Quek, "A hybrid dqn and optimization approach for strategy and resource allocation in mec networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 7, pp. 4282–4295, 2021.

[22] J. Kim, J. Lee, T. Kim, and S. Pack, "Deep q-network-based cloud-native network function placement in edge cloud-enabled non-public networks," *IEEE Trans. Netw. Service Manag.*, pp. 1–1, 2022.

[23] L. Cui, E. Ni, Y. Zhou, Z. Wang, L. Zhang, J. Liu, and Y. Xu, "Towards real-time video caching at edge servers: A cost-aware deep q-learning solution," *IEEE Trans. Multimedia*, pp. 1–1, 2021.

[24] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *arXiv preprint arXiv:1610.03295*, 2016.

[25] S. Arora, P. A. Frangoudis, and A. Ksentini, "Exposing radio network information in a mec-in-nfv environment: the rnisaas concept," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, 2019, pp. 306–310.

[26] G. Avino, P. Bande, P. A. Frangoudis, C. Vitale, C. Casetti, C. F. Chiasserini, K. Gebru, A. Ksentini, and G. Zennaro, "A MEC-based extended virtual sensing for automotive services," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1450–1463, 2019.

[27] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, "srsLTE: An open-source platform for LTE evolution and experimentation," in *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, 2016, pp. 25–32.

[28] Z. Li, A. Aaron, I. Katsavounidis, A. Moorthy, and M. Manohara, "Toward a practical perceptual video quality metric," *The Netflix Tech Blog*, vol. 6, no. 2, 2016.

[29] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 187–198.

[30] *3GPP TS 23.501 V16.3.0 Technical Specification Group Services and System Aspects; System architecture for the 5G System (5GS); Stage 2, (Release 16)*, 3GPP, 12 2019.

[31] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[32] A. A. Sherstov and P. Stone, "Function approximation via tile coding: Automating parameter choice," in *Abstraction, Reformulation and Approximation*, J.-D. Zucker and L. Saitta, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 194–205.

[33] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *International Conference on Autonomous Agents and Multiagent Systems*. Springer, 2017, pp. 66–83.

[34] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *IEEE INFOCOM*, 2019, pp. 514–522.

[35] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *IEEE INFOCOM*, 2019, pp. 10–18.

[36] X. Ma, A. Zhou, S. Zhang, and S. Wang, "Cooperative service caching and workload scheduling in mobile edge computing," in *IEEE INFOCOM*, 2020, pp. 2076–2085.

[37] F. Wang, C. Zhang, F. wang, J. Liu, Y. Zhu, H. Pang, and L. Sun, "Intelligent edge-assisted crowdcast with deep reinforcement learning for personalized qoe," in *IEEE INFOCOM*, 2019, pp. 910–918.

[38] Q. Liu, S. Huang, J. Opadere, and T. Han, "An edge network orchestrator for mobile augmented reality," in *IEEE INFOCOM*, 2018, pp. 756–764.

[39] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *IEEE INFOCOM*, 2020, pp. 257–266.

[40] Y. He, J. Ren, G. Yu, and Y. Cai, "Optimizing the learning performance in mobile augmented reality systems with CNN," *IEEE Transactions on Wireless Communications*, vol. 19, no. 8, pp. 5333–5344, 2020.

[41] R. Combes and A. Proutiere, "Dynamic rate and channel selection in cognitive radio systems," *IEEE J. Sel. Areas Commun.*, vol. 33, no. 5, pp. 910–921, May 2015.

[42] R. Combes, J. Ok, A. Proutiere, D. Yun, and Y. Yi, "Optimal rate sampling in 802.11 systems: Theory, design, and implementation," *IEEE Trans. Mobile Comput.*, vol. 18, no. 5, pp. 1145–1158, May 2019.

[43] H. Gupta, A. Eryilmaz, and R. Srikant, "Low-complexity, low-regret link rate selection in rapidly-varying wireless channels," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 540–548.

[44] J. Ma, T. Nagatsuma, S. Kim, and M. Hasegawa, "A machine-learning-based channel assignment algorithm for iot," in *Proc. Intl. Conf. Artificial Intell. Informat. Commun. (ICAIIC)*, 2019, pp. 1–6.

[45] S. Hasegawa, S. Kim, Y. Shoji, and M. Hasegawa, "Performance evaluation of machine learning based channel selection algorithm implemented on iot sensor devices in coexisting IoT networks," in *Proc. IEEE Consumer Commun. Netw. Conf. (CCNC)*, 2020, pp. 1–5.

[46] M. A. Qureshi and C. Tekin, "Fast learning for dynamic resource allocation in AI-enabled radio networks," *IEEE Trans. Cogn. Commun. Netw.*, vol. 6, no. 1, pp. 95–110, Mar. 2020.

[47] I. Comșa, S. Zhang, M. E. Aydin, P. Kuonen, Y. Lu, R. Trestian, and G. Ghinea, "Towards 5G: A reinforcement learning-based scheduling solution for data traffic management," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 4, pp. 1661–1675, Dec. 2018.

[48] I. Comșa, R. Trestian, G. Muntean, and G. Ghinea, "5MART: A 5G SMART scheduling framework for optimizing QoS through reinforcement learning," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 2, pp. 1110–1124, June 2020.

[49] S. Tripathi, C. Puligheddu, C. F. Chiasserini, and F. Mungari, "A context-aware radio resource management in heterogeneous virtual rans," vol. 8, no. 1, 2022, pp. 321–334.

[50] L. Baldesi, F. Restuccia, and T. Melodia, "Charm: Nextg spectrum sharing through data-driven real-time o-ran dynamic control," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, 2022.

[51] S. Tripathi, C. Puligheddu, S. Pramanik, A. Garcia-Saavedra, and C. F. Chiasserini, "VERA: Resource orchestration for virtualized services at the edge," in *2022 IEEE International Conference on Communications (ICC)*, 2022.

Sharda Tripathi received the Ph.D. degree in Electrical Engineering from Indian Institute of Technology Delhi, Delhi, India in 2019. She is currently serving as an Assistant

Professor in the Department of Electrical & Electronics Engineering at Birla Institute of Technology and Science Pilani, Pilani campus, India. Her research interests are in the design and analysis of 5G networks and beyond.

Corrado Puligheddu (M'20) received his MSc degree in 2019 from Politecnico di Torino, Italy, where he is currently a PhD student. His research interests are in wireless networks and mobile applications.

Somreeta Pramanik received the M.S. degree from Indian Institute of Technology, Kharagpur in 2018. She is currently pursuing the Ph.D. degree in Politecnico di Torino, Italy. Her research interest is in wireless networks and 5G communication system.

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 until 2015. Currently, he is a Principal Researcher at NEC Laboratories Europe. His research interests lie in the application of fundamental mathematics to real-life wireless communication systems.

Carla Fabiana Chiasserini (F'18) worked as a visiting researcher at UCSD and as a Visiting Professor at Monash University in 2012 and 2016. She is currently a Professor at Politecnico di Torino and EiC of Computer Communications.