PRORL: Proactive Resource Orchestrator for Open RANs using Deep Reinforcement Learning

Alessandro Staffolani, Victor-Alexandru Darvariu, Luca Foschini, Michele Girolami, Paolo Bellavista and Mirco Musolesi

***Abstract*—Open Radio Access Network (O-RAN) is an emerging paradigm proposed for enhancing the 5G network infrastructure. O-RAN promotes open vendor-neutral interfaces and virtualized network functions that enable the decoupling of network components and their optimization through intelligent controllers. The decomposition of base station functions enables better resource usage, but also opens new technical challenges concerning their efficient orchestration and allocation. In this paper, we propose Proactive Resource Orchestrator based on Reinforcement Learning (PRORL), a novel solution for the efficient and dynamic allocation of resources in O-RAN infrastructures. We frame the problem as a Markov Decision Process and solve it using Deep Reinforcement Learning; one relevant feature of PRORL is that it learns demand patterns from experience for proactive resource allocation. We extensively evaluate our proposal by using both synthetic and real-world data, showing that we can significantly outperform the existing algorithms, which are typically based on the analysis of static demands. More specifically, we achieve an improvement of 90% over greedy baselines and deal with complex trade-offs in terms of competing objectives such as demand satisfaction, resource utilization, and the inherent cost associated with allocating resources.**

***Index Terms*—O-RAN, reinforcement learning, resource allocation, multi-objective optimization**

I. Introduction

ONE of the aims of the fifth-generation (5G) cellular network infrastructure is to provide very high data rates with extremely low latency and Quality of Service (QoS) improvements for the final users. In response to these challenges, providers have implemented new technologies such as massive Multiple Input, Multiple Output (MIMO) [1], millimeter wave and sub-terahertz communications [2], network-based sensing [3], virtualization through Network Functions Virtualization (NFV) and Software-Defined Networking (SDN) [4], and Machine Learning (ML)-based digital signal processing [5], among others. These solutions enhance network capabilities but also come at the expense of increased management complexity and cost for the operators. To mitigate this complexity and avoid vendor lock-in, the Open-RAN (O-RAN) Alliance [6] has defined and standardized open interfaces in order to decouple hardware from software for enhanced flexibility and to enhance the support and usage of AI solutions for network operations and management.

O-RAN extends the 3GPP NR 7.2 split for base stations [7], which disaggregates their functions into a Central Unit (CU), a Distributed Unit (DU), and a Radio Unit (RU) (called O-CU, O-DU, and O-RU respectively in the O-RAN specifications). CUs and DUs are virtualized on-edge cloud servers, while RUs are deployed at cell sites. Moreover, O-RAN connects base station functionalities to intelligent controllers, also known as RAN Intelligent Controllers (RICs), which have visibility of network performance indicators and are utilized to aggregate Key Performance Measurements (KPMs) for supporting closed-loop control applications for the overall infrastructure. In the O-RAN specifications, these controllers are categorized as near-real-time RICs and non-real-time RICs [8].

The advantages of the O-RAN architecture are many: it allows for better usage and management of resources thanks to virtualization, centralization, and dynamic reallocation; it can reduce management costs and generate significant energy savings; and it enables the potential augmentation of network capacity through the addition of virtual resources to its logically centralized pool.

In this paper, we present Proactive Resource Orchestrator based on Reinforcement Learning (PRORL), a novel solution for *dynamic orchestration and management of resources in O-RAN*. PRORL is an adaptive learning-based orchestrator that learns patterns in the demand from experience and uses them to proactively allocate resources by considering the expected effect over future time windows.[1]

More specifically, we target a three-tier network infrastructure with O-RAN components (see Figure 1): at the highest level (*Regional Cloud*), we place the resource pool, which is directly connected to the core network in a regional cloud data center. In the middle (*Edge Cloud*), the infrastructure hosts edge data centers, also called Points of Presence (PoPs), which communicate through the O1 interface [9] to the Regional Cloud: PoPs are responsible for sharing the resources received from the upper tier to the CUs, DUs and the lower tier, thanks to CU's Radio Resource Control (RRC) [10] and Service Data Adaptation Protocol (SDAP) [11] layers, which manage the connection lifecycle and the Quality of Service of the traffic flows (also known as bearers). The last tier (*Cell Site*) consists of RUs that receive capacity from DUs of their PoP. PRORL is strongly original in its proposal of a learning-

Manuscript received July 17, 2023; revised January 30, 2024.

Alessandro Staffolani, Luca Foschini, Paolo Bellavista and Mirco Musolesi are with the Department of Computer Science and Engineering, University of Bologna, Bologna, Italy. E-mail: alessandro.staffolani, luca.foschini, paolo.bellavista, mirco.musolesi@unibo.it.

Alessandro Staffolani and Michele Girolami are with the National Council of Research Italy, ISTI-CNR, Pisa, Italy. E-mail: alessandro.staffolani, michele.girolami@isti.cnr.it.

Victor-Alexandru Darvariu and Mirco Musolesi are with the Department of Computer Science, University College London, London, UK. E-mail: v.darvariu, m.musolesi@ucl.ac.uk.

[1]Note that for the actual virtualized resource movement and placement, PRORL integrates with external and existing Network Functions Virtualization (NFV) management frameworks.

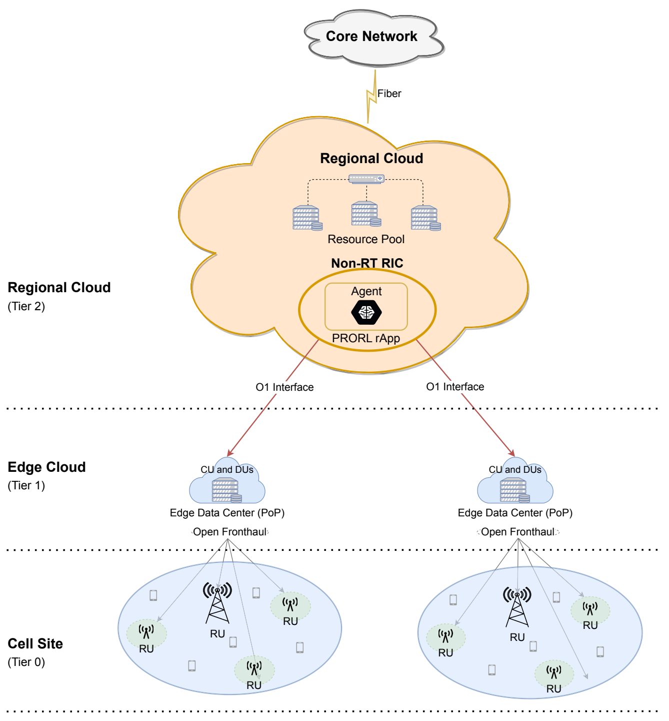

Fig. 1. Graphical overview of our reference multi-tier O-RAN architecture.

based orchestrator installed on the Regional Cloud as an rApp that controls the pool of currently available resources and is responsible for suitably allocating them to the PoPs. It is worth mentioning that the O-RAN PRORL rApp is a plug-and-play component that implements custom logic in the O-RAN ecosystem. It directly communicates with non-RT RICs to collect network-related KPMs and to inject resource management policies.

In comparison with the currently deployed O-RAN-related solutions, we strongly believe that PRORL advances the state-of-the-art in terms of effective and efficient use of resources. In fact, current base stations are usually provided with sufficient resources for accommodating peak hours, which then remain over-provisioned for the rest of the day [12]. On the contrary, our orchestrator can react to rapid changes in loads and, simultaneously, proactively move resources while maintaining an adequate QoS. In this way, it can also reduce the total capacity required in the system and the Operating Expenditure (OPEX) for reallocating the resources between PoPs.

Design Challenges. In order to design PRORL, we addressed a series of challenges related to resource allocation:

1) *Allocation strategy*: an effective and efficient strategy is required for orchestrating resources among PoPs. The strategy has to select PoPs that will receive additional resources taken either from the pool or from other nodes that have exceeding capacity with respect to their current demand. It also has to optimize PoP satisfaction (by allocating sufficient capacity to them) and, simultaneously, to avoid wasting resources by over-provisioning.
2) *Adaptability to demand dynamics*: demand varies over space and time due to the variety of users, applications, and services that use the network. The solution has to address demand dynamics in a proactive manner in order to move resources before they are needed (*proactivity of the strategy*). Furthermore, the enforced strategy must be able to react quickly to unexpected changes (*reactivity of the strategy*).
3) *Movement cost*: moving resources carries an inherent cost due to the de-activation and re-allocation of resources on different edges of the network deployment environment. Thus, the allocation strategy needs to take into account the number of resource movements.

Our Contributions. The main contributions of this paper can be summarized as follows:

1) We model the capacity allocation problem as a decision-making process in which PoPs have to serve connectivity to the underlying O-RAN components. An agent has to move resources from the centralized pool to the PoPs according to their aggregated demand; the agent decision is guided by a numerical reward signal based on an objective function to be maximized. We design our original objective function in order to balance multiple competing objectives weighting their effects; namely, we optimize system satisfaction, resource utilization, and the cost associated with resources movement (the OPEX). We formulate the problem in the Markov Decision Process (MDP) framework and solve it using Deep Reinforcement Learning. Our novel model architecture features a decomposition of the action space, which allows for a substantially faster convergence of the agent.
2) We extensively evaluate our solution using a publicly available dataset composed of Call Detail Records (CDRs) collected in the city of Milan for two months in 2013, which are considered a solid use case in the related literature [13]. We also analyze the effect of different trade-offs among the three objectives that PRORL can optimize in order to show the flexibility of the proposed approach. Finally, we conduct a sensitivity analysis using realistic synthetic data to experimentally study the robustness of the model when dealing with challenging scenarios, such as immediate demand peaks or system overloading.

Main Results. Our experimental evaluation over both real and synthetic data shows that PRORL can achieve significantly better performance in comparison with baseline greedy solutions. Moreover, our approach can effectively deal with complex trade-offs in terms of competing objectives. PRORL is flexible enough to outperform the baselines over multiple trade-off configurations, with an average improvement of 90% over them. We also prove the robustness of our approach, evaluating it on 4 different challenging scenarios using synthetic data. The results confirm the superiority of PRORL in all considered scenarios, with improvements in single objectives of up to $10\times$ the performance of the baselines.

II. Related Work

The problem of resource optimization in RAN can be divided into low-level and high-level resource allocation problems. The former addresses allocation at levels close to the

TABLE I
SUMMARY OF THE DIFFERENCES BETWEEN OUR WORK AND HIGH-LEVEL O-RAN RESOURCE ALLOCATION STATE-OF-THE-ART.

Citation	Description	Allocation Algorithm	Deployment	Optimization Horizon	Real Data Evaluation
[14]	Two-step system: forecasting model to accurately model and predict the traffic volume and handover count, and constrained optimization problem.	Multivariate Long Short Term Memory (MuLSTM) model for the forecasting step, Resource-Constrained Label-Propagation (RCLP) algorithm for the optimization problem.	C-RAN environments.	One-step look ahead based on forecast prediction.	✓
[15]	Reinforcement Learning approach to map the split between CUs and DUs functionalities in order to optimize the energy consumption in green O-RAN.	Q-learning and SARSA.	Green O-RAN environments.	The agent optimizes the objective over long time horizons.	✓
[16]	An algorithm to improve User Equipment (UE) placement taking into account radio quality, bandwidth, and user distribution.	Hand-crafted heuristic algorithm	O-RAN environments.	Current step.	✗
[17]	Deep Reinforcement Learning approach to solve radio resource slicing and priority-based core network slicing in order to optimize Spectral Efficiency, Quality of Experience, and Service Function Chain execution time.	DQN.	Network Slicing.	The agent optimizes the objective over long time horizons.	✗
[18]	A collaborative learning framework that combines Deep Learning and Reinforcement Learning with the goal of improving resource-multiplexing gain among slices while meeting specific service requirements for radio access network (RAN) slices.	LSTM for traffic prediction, A3C for the scheduling decision.	Network Slicing in RAN environments.	The agent optimizes the objective over long time horizons.	✗
PRORL	PRORL frames the problem of dynamic orchestration and management of resources in O-RAN as a MDP in which a centralized agent learns patterns of demand to move resource units from the pool to the PoPs and vice versa, in order to optimize multiple competing objectives.	Double DQN with prioritized experience replay and dueling network, using a split design to reduce action space dimension.	O-RAN environments.	The agent optimizes the objective over long time horizons.	✓

physical layer, such as spectral efficiency, radio resources, and power allocation, which typically involves control of RUs and sometimes DUs. The latter addresses orchestration problems further up the stack such as deployment, cell selection, and CU/DU control by activation and deactivation. Our proactive orchestrator belongs to the high-level resource allocation class. To the best of our knowledge, PRORL is the first attempt to address the problem of orchestrating resources among different pools while optimizing multiple conflicting objectives. Furthermore, it considers the effect of allocations over multiple timesteps, during which the demand naturally varies, and it is evaluated on both synthetic and real-world data.

Low-Level O-RAN Resource Allocation. In the context of Heterogeneous Cloud Radio Access Networks (H-CRANs), Peng et al. propose two optimization problems [19], [20]. In the first work, the authors mitigate inter-tier interference while optimizing energy efficiency by assigning resource blocks and transmission power. The optimization problem is formulated as a non-convex fractional program, which is solved by means of the Lagrange dual decomposition. The second work focuses on the impact of the cost of different types of fronthaul in C-RAN. The authors propose a joint optimization of energy efficiency and the wired/wireless fronthaul cost. The problem is formulated as a non-convex beamformer problem that constrains fronthaul capacity and transmission power. It is solved by developing an outer-inner loops algorithm. In the outer loop, the primal problem is transformed into an equivalent sub-problem using the bisection search method while, in the inner loop, the sub-problem is solved by the weighted minimum mean squared error approach. In the O-RAN context, Wang et al. [21] study the RU-DU resource assignment problem in O-RAN. They model the problem as a 2D bin packing problem and present a deep reinforcement learning agent with Monte Carlo tree search to solve the problem. They evaluate their solution both on synthetic and real-world data, discovering improved policies for resource assignment in diverse network conditions. In [22], a reinforcement learning solution dynamically adapts the per-flow resource allocation, modulation, and coding scheme in order to satisfy the traffic flow requirements. In [23], reinforcement learning methods are proposed to manage sessions for ultra-reliable and low latency communication (URLLC).

High-Level O-RAN Resource Allocation. In [24] and [14], Chen et al. propose a two-step system for mapping virtualized resources in H-CRAN environments. A traffic forecasting model is used for predicting the demand in the next time window. The authors define a constrained optimization problem to find the best mapping given the predicted demand. They evaluate both approaches using real-world data. However, even if the mapping of resources employed by the authors is conceptually similar to that presented in our work, these solutions do not consider the effect of the demand variation over subsequent time intervals, relying instead on the accuracy of the next prediction. Similarly, in [15], Pamuklu et al. propose a reinforcement learning solution for mapping the split of functionalities between CUs and DUs in Green O-RAN, where the objective is to reduce energy consumption while utilizing renewable energy sources. In [16], the authors propose an algorithm to improve User Equipment (UE) placement taking into account radio quality, bandwidth, and user distribution. Finally, several works address the problem of Network Slicing using deep reinforcement learning [17], [18], [25]–[28], which involves the allocation of resources to create network partitions where different policies and QoS requirements can be met. In contrast, our approach aims to support a finer level of granularity by orchestrating the pool of resource units.

III. BACKGROUND

In this section, we briefly introduce the theoretical framework used in this work. Firstly, we introduce the classic single-objective reinforcement learning problem. Then, we present the challenges of optimizing multiple objectives, as is the case

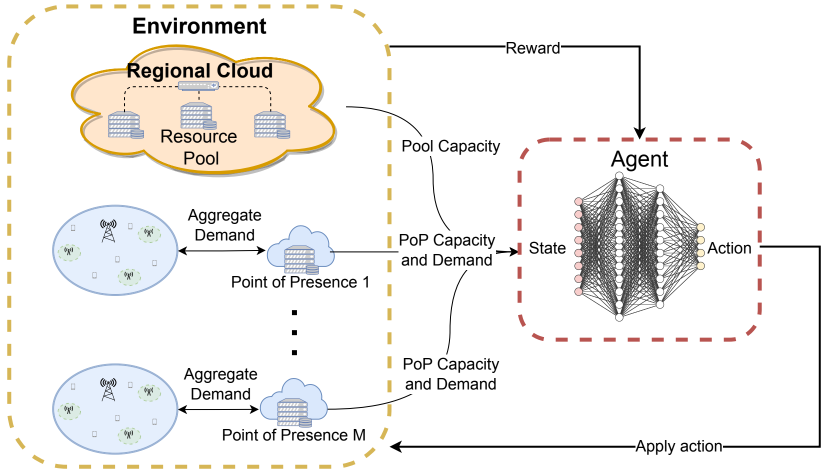

Fig. 2. PRORL at a glance. The environment is composed by the resource pool and PoPs. A state is derived from the environment representing the demand and capacity of the nodes and the pool. The state is given as input to the agent that outputs the action to be taken. The actions consist of a movement of resource units, which results in a new configuration.

in the problem setting of PRORL. Finally, we provide an in-depth discussion about the implementation of the proposed algorithm.

Single-Objective MDP. Markov Decision Processes are used to model a learning process based on interactions [29]. The learner and decision-maker is usually called an *agent*. The agent interacts with the environment (in our case the O-RAN system) at discrete time steps $t = 0, 1, 2, ..., T$. At each time step t, the agent receives a representation of the environment (defined as a *state*) $S_t \in \mathcal{S}$ and selects an *action* $A_t \in \mathcal{A}(S_t)$. $\mathcal{S}$ is the set of the possible states and $\mathcal{A}(S_t)$ is the set of possible actions in state S_t. As a consequence of the action, the agent receives a *reward* R_{t+1} and enters a new state S_{t+1}. The goal of the agent is to define a policy $\pi(A_t|S_t)$, a probability distribution over actions for a given state, which maximizes the discounted return:

$$G_t = \sum_{k=0}^{T} \gamma^k R_{t+k+1} \tag{1}$$

where R_{t+k+1} is the reward at time $t+k+1$ and γ a discount rate with $0 \leq \gamma \leq 1$. We then define the *state-value function* given a policy π as $V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$, which is the expected discounted return when following the policy π from the state s onwards. We also define the *action-value function* given policy π as $Q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$, i.e., the expected discounted return when taking action a in state s then following the policy π. In deep reinforcement learning, the policy π is typically parametrized using a deep neural network with parameters θ.

Multi-Objective MDP. The Multi-Objective Markov Decision Process (MOMDP) captures problems with more than one objective to be optimized. The only difference compared to the standard MDP is that, instead of a scalar, the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ returns a vector of rewards, one for each objective, with $d \geq 2$ indicating the number of objectives [30]. Therefore, the value of a state $\mathbf{V}_\pi(s) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k \mathbf{R}_{t+k+1}|S_t = s]$ given a policy π is also a vector (please note that we use bold fonts for indicating vectors). In contrast to single-objective MDPs, in MOMDP we cannot define a natural order between different policies without any additional information about how to prioritize them, because the value of a policy can be ambiguous. A typical method for finding solutions for a MOMDP is to use the *utility-based* approach. We define a *utility function* (also known as *scalarization function*) that projects the multi-objective value $\mathbf{V}_\pi$ to a scalar value:

$$V_\pi^{\mathbf{w}}(s) = \mathcal{U}(\mathbf{V}_\pi(s), \mathbf{w}) \tag{2}$$

where $\mathbf{w}$ is the weight vector parametrizing $\mathcal{U}$. The scalarization can be either linear or non-linear and, depending on this choice, we can find different solutions to the problem. For simplicity, in this work, we assume that a linear combination of the objectives is sufficient to capture the trade-offs between the different desiderata. The full formulation of the linear utility function is given in Section IV, Equation 9.

Decision-Making Algorithm. The choice of a linear utility function gives the added benefit that any single-objective RL algorithm is applicable. Therefore, we use the DQN [31], [32], which is an off-policy value-based method that is more data-efficient with respect to on-policy methods. This last advantage is a critical aspect in a real and complex environment such as that of O-RANs, where data acquisition is expensive.

DQN uses a function approximator (typically a deep neural network) for parametrizing, with weights θ, the action-value function $\hat{Q}(s, a, \theta)$. Further to tabular Q-learning, DQN leverages the experience replay and the target network $\hat{Q}^-$. The former [33] consists of storing tuples of agent's experience $\langle S_t, A_t, \mathbf{R}_t, S_{t+1}\rangle$ in a replay buffer $\mathcal{D}$ and sampling a mini-batch at each learning step. It allows for better data efficiency and reduces the variance in the updates. The latter makes the DQN more stable by copying the weights θ of the online network to those of the target network θ^- every C steps.

In the past years, the research community has proposed several improvements to the original DQN algorithm [34], some of which we use in our implementation. In particular, we adopt the Double DQN [35] updates, which are able to

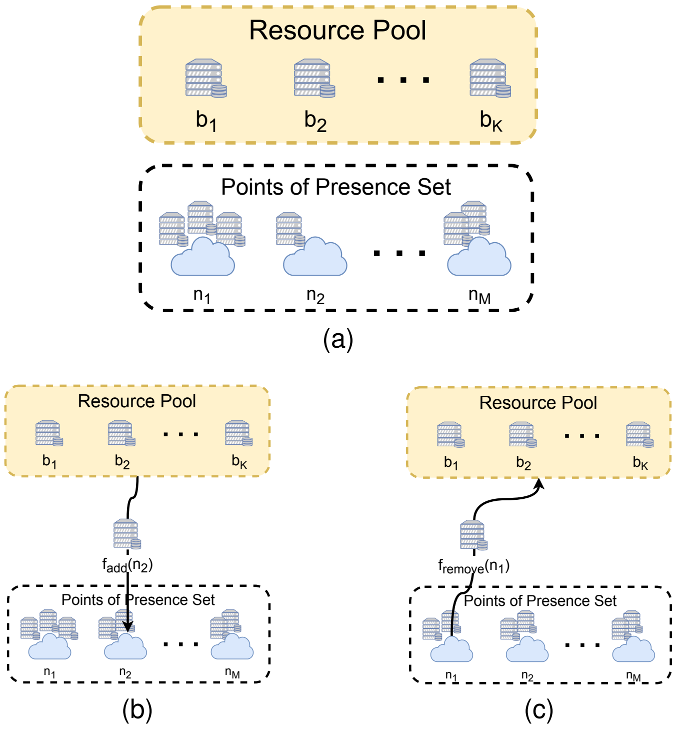

Fig. 3. (3a) Graphical representation of the pool $\mathcal{B}$ and the PoPs set $\mathcal{N}$. (3b) Example of movement following the execution of f_{add}. (3c) Example of movement following the execution of f_{remove}.

deal with the problem of the overestimation of Q-values of the original approach. We also adopt *prioritized experience replay* [36], which, instead of sampling uniformly from the replay buffer, samples entries proportionally to a weight, which is a function of the Temporal Difference (TD) estimation error, hence prioritizing "important" transitions. We opt for the variant of PER that determines weights corresponding to the last encountered absolute TD error, which has been shown to be highly performant yet computationally efficient. Finally, we use the *dueling network* approach proposed by Wang et al. [37], which modifies the Q-network architecture by splitting the last layer into two branches that output the advantage and value respectively. The two branches are subsequently factorized in order to obtain the Q-values in output. Indeed, this technique allows for better generalization across the actions.

IV. Resource Orchestration with PRORL

In this section, we present the design and implementation of PRORL, our resource orchestrator solution, illustrated in Figure 2. We firstly describe the associated MDP starting from the concepts introduced in the previous section. Practically, the agent uses the system status as input and outputs the action (a reconfiguration of resources) that is applied to the network deployment environment, which is composed of the resources pool and the PoPs. The environment provides the agent with the new status and the reward associated with the allocation.

A. Problem Formulation

Formally, we are given a resource pool $\mathcal{B} = \{b_1, ..., b_K\}$ initialized with K units, each of size σ and a set of PoPs $\mathcal{N} = \{n_1, ..., n_M\}$ (also referred to as nodes) of cardinality M, where $K \gg M$, as shown in Figure 3a. Each node n_i is described by the tuple $\langle d_{t,n_i}, c_{t,n_i} \rangle$. The *demand* $d_{t,n_i} \in \mathbb{R}$ represents the load on the node n_i at time t obtained by aggregating all the demand in the controlled area, while the *capacity* $c_{t,n_i} \in \mathbb{N}$ represents the number of resource units allocated to node n_i. We note that the system is able to control the allocated capacities, while the demands are external.

The system allows performing two resource unit movements (i.e., sub-actions) per time step t. The former, $f_{add}(n_{add})$, allows moving one resource unit from the pool $\mathcal{B}$ to a PoP n_{add}, illustrated in Figure 3b. The latter, $f_{remove}(n_{remove})$, allows releasing one resource unit from a PoP n_{remove}, with $n_{add} \neq n_{remove}$, illustrated in Figure 3c. Moving a resource unit from the pool means activating and allocating it into its new location. Conversely, releasing a resource unit means deactivating and moving back to the pool. Every time a resource unit is allocated or released a cost κ is incurred. In addition, for each movement, the system allows waiting (performing no movement), which incurs a cost $\kappa = 0$.

We define $\Delta_{t,n_i} = c_{t,n_i} - d_{t,n_i}$ as the difference between the capacity and demand for node n_i at time t. Given a target τ, our objective is:

$$\min \sum_{t=0}^{T} \left(\sum_{i=0}^{M} c_{t,n_i} \right) \tag{3}$$

$$+ \kappa \mathbb{1}(A_t^1 \neq wait) + \kappa \mathbb{1}(A_t^2 \neq wait) \tag{4}$$

$$\text{s.t. } \Delta_{t,n_i} \geq \tau, \quad \forall n_i \in \mathcal{N}, \tag{5}$$

where $\mathbb{1}$ is the indicator function, which is equal to 1 if the condition expressed as argument is true and 0 otherwise, while A_t^1 and A_t^2 are the two sub-actions, respectively. Intuitively, over a time horizon of T steps, we need to minimize the allocated capacities (first term of the sum) and the cost of movements (second and third term) while satisfying the demands requested by the nodes (the constraint).

The presented problem shares significant similarities with the field of Inventory Management [38], where it is necessary to determine the quantity of *stock* (capacity) allocated to each *warehouse* (node). In this context, the Dynamic Capacitated Lot-Sizing problem is recognized as NP-hard [39]. In addition, also the Unit Commitment problem in electrical power production shares similarities and is known to be NP-hard [40], [41]. Hence, deriving the optimal solution to large instances of the problem is intractable, particularly given the hidden nature of demands, necessitating adjustments to accommodate its variability. Due to these challenges, we posit that an approach like PRORL, providing approximate solutions, is needed.

In order to simplify the presentation, without loss of generality, we set $\tau = 0$ and the time step size t to 1 hour for the remainder of the paper. The time step size influences the resolution at which the demand is collected and evaluated. The choice of the time interval is of key importance. If it is too short, the changes might be temporary and, in general, very noisy. If it is too long, the algorithm will not be able to adapt to non-negligible demand variations that might happen between the two sampling points. Figure 5 shows that, for the considered real-world dataset, this is an appropriate level of

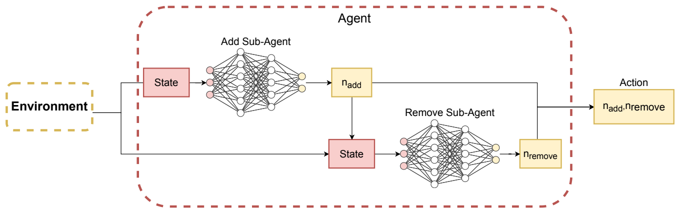

Fig. 4. PRORL agent decomposition into two sub-agents.

granularity to consider. It is worth noting that, given a scenario with different demand variability, our approach can support a significantly smaller granularity that approaches real time. In general, using a learned model in such optimization problems may be appropriate when decisions need to be made rapidly and running a solver is unfeasible [42].

B. Definition of the MDP

We now present the formulation of the orchestration problem underlying PRORL as an MDP, discussing the action and state spaces and the chosen reward structure.

Actions and Decomposition of the Action Space. The agent is a logically centralized component that receives the current environment state as input, selects actions and improves its policy in order to maximize the reward received. In this work, we split the agent in two components in order to reduce the dimension of the action space. The actions consist of either waiting or adding (removing) resources from one of the M nodes. Therefore, the combined action space has dimension $(M+1)^2$. On the other hand, if we decompose the action into two independent sub-actions, the overall action space dimension becomes $(M+1)+(M+1) = 2(M+1)$. A smaller action space allows for a quicker learning process given the reduction of its size, as experimentally demonstrated in the evaluation section.

We therefore implement two independent agents that act sequentially as depicted in Figure 4. We refer to them as the *add* and *remove* sub-agents, respectively. Furthermore, before each step, the environment shrinks the two action spaces if some criteria are met, in order to prevent invalid actions. For the *add* sub-action, the space is reduced to contain only the wait action if the current capacity of the pool $\mathcal{B}$ is zero. Instead, for the *remove* sub-action we disable the index corresponding to the node selected as n_{add}, if it is not the *wait* action, and we disable nodes with zero allocated capacity.

States. The state should encapsulate all the relevant information about the environment that is useful to the agent for selecting actions. More specifically, for our resource orchestration problem, it must capture the capacity of the nodes and the pool, as well as the demand in different locations. The state of each sub-agent contains the following information:

- *Pool capacity*: the current dimension of the pool $\mathcal{B}$. This feature tells the agent the number of currently unallocated resource units that can be moved to the nodes.
- *Node capacity*: a vector of dimension M with one entry representing the current capacity c_{t,n_i} for each node n_i.
- *Node demand*: a vector of dimension M with the current demand d_{t,n_i} for each node n_i.

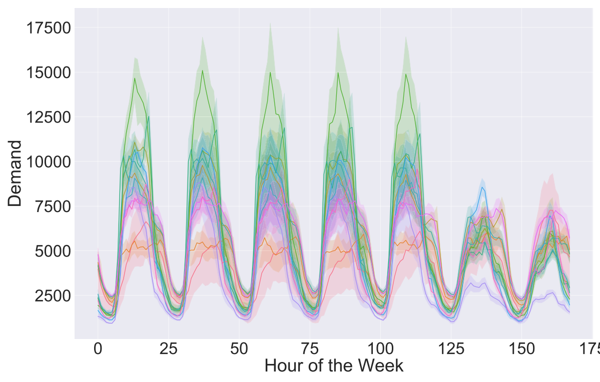

Fig. 5. Demand variation over 8 weeks for the nodes selected for evaluating PRORL. The data are from the network traces [13] gathered in the city of Milan in 2013.

- *Capacity surplus*: a vector of dimension M, whose elements are the current differences between capacity and demand Δ_{t,n_i} for each node n_i. This feature, although in principle redundant (since it is a linear combination of other features), is an inductive bias that has proven beneficial in preliminary experiments.
- *Current time*: a one-hot encoding of the hour and the day of the week of the current time step. This feature lets the agent associate patterns in demands with a time interval, in effect capturing seasonality in the data.
- *Remaining attempts*: it is a scalar value that tells the agent the number of attempts remaining in the current episode (please refer to the Trajectory Pruning paragraph for additional details).

The state of the *remove* sub-agent contains an additional piece of information, namely, the one-hot encoding of the sub-action selected by the *add* sub-agent, i.e., the index of the selected node or a vector of zeros if *wait* is selected. The *add* sub-action feature is used by the *remove* sub-agent for coordination. Furthermore, all the state features are normalized in order to have values between 0 and 1. Moreover, for the demand-related features, we also divide the values by the resource unit capacity and apply the ceiling function $\lceil x \rceil$. By doing so, we transform the demand into an integer value and we reduce the number of possible states, which leads to a more efficient learning process.

Rewards. The reward signal is used to discriminate between beneficial and non-beneficial actions. The quality of the decision does not directly impact the placement of a single service, since our solution governs the allocation of resource units at the underlying Points of Presence (PoPs), which are then responsible for ensuring the Quality of Service (QoS) for individual users. We define three different reward signals, one for each *component* of the objective. Therefore, at each time step t, the vector of rewards is $\mathbf{R}_t = \{R_t^{\text{surplus}}, R_t^{\text{cost}}, R_t^{\text{gap}}\}$.

The first component, corresponding to term 3, measures the unused resources, which are essentially wasted. We define the *surplus* as the number of unused units on every node in order minimize the units allocated on all the nodes:

$$R_t^{\text{surplus}} = -\sum_{i=1}^{M} \Delta_{t,n_i} \mathbb{1}(\Delta_{t,n_i} > \tau) \tag{6}$$

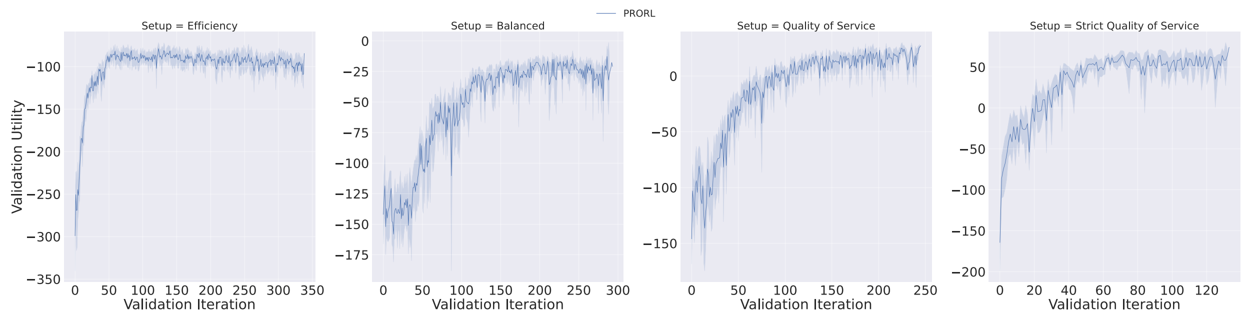

Fig. 6. PRORL validation total utility during training for the four setups (higher is better). Note that, while the number of episodes within an iteration may differ due to the trajectory pruning mechanism, the rightmost point in all the four plots corresponds to the same 336000 steps.

where $\mathbb{1}$ is the indicator function, which is equal to 1 if the condition expressed as argument is true, 0 otherwise. The second and third components of term 4 are the costs associated with resource movements. We use the following reward:

$$R_t^{\text{cost}} = -\kappa \mathbb{1}(A_t^1 \neq wait) - \kappa \mathbb{1}(A_t^2 \neq wait) \tag{7}$$

where A_t^1 and A_t^2 are the two sub-actions respectively.

The final component (corresponding to the constraint 5) measures the level of satisfaction of the nodes. The *remaining gap* is defined as the number of units that would be necessary in order to satisfy the constraint for every node. The corresponding reward signal at time step t is:

$$R_t^{\text{gap}} = -\sum_{i=1}^{M} \Delta_{t,n_i} \mathbb{1}(\Delta_{t,n_i} < \tau) \tag{8}$$

Utility Function. The utility function is used for composing the rewards that drive the learning process. As discussed in Section III, we adopt the following linear utility function:

$$\mathcal{U}(\mathbf{R}_t) = w_{\text{surplus}}\phi(R_t^{\text{surplus}}) + w_{\text{cost}}\phi(R_t^{\text{cost}}) + w_{\text{gap}}\phi(R_t^{\text{gap}}) \tag{9}$$

where $w_{surplus} + w_{cost} + w_{gap} = 1$. The values of the weights are set according to the requirements of the designer of the system. Different sets of values will lead to different trade-offs, an aspect that is discussed in Section VI. Additionally, $\phi(R)$ is a normalization function applied to the rewards before combining them. We normalize the rewards in order to weigh values with the same magnitude. More specifically, the goal is to obtain values in the range $[0, 1]$, where 0 corresponds to the worst value and 1 to the optimal one.

Trajectory Pruning. We also implement a mechanism for pruning sub-optimal trajectories during the training of our model. At initialization time, the environment starts with D *attempts* and, after each time step, we compute the number of unsatisfied nodes, i.e., the number of nodes with $\Delta_{t,n_i} < \tau$. If the number of nodes for which the demand is not satisfied is greater than 0, an attempt is lost. When the environment reaches zero attempts, the episode terminates. This has the effect of discarding clearly sub-optimal trajectories, encouraging the agent to maintain longer episodes during which higher returns can keep being received. We note that this technique is also called "early termination" in some RL works.

Computational Complexity. For performing inference, our solution involves interacting with two sub-agents at every time step, as depicted in Figure 4. The interaction requires building the representations of the two states, applying the action shrinking in order to disable unavailable actions, and performing the forward passes of the policies of the sub-agents. Assuming that the sizes of the neural networks remain constant in terms of number of layers and number of units in the hidden layers, each of the steps has complexity $\mathcal{O}(M)$. Therefore, the overall computational complexity of our approach is $\mathcal{O}(M)$, i.e., linear in the number of nodes. The training cost cannot be expressed analytically given the dependence on the number of steps to convergence, which is challenging to determine in reinforcement learning with neural networks for function approximation. Therefore, we assess convergence empirically by studying the validation performances, as shown in Figure 6. In practice, the models take approximately 5 hours to train for the real-world data case on a single Nvidia RTX A5000 GPU. Above all, it is worth noting that training represents a one-off cost, and retraining the network might become necessary only if there is a substantial change in demand patterns.

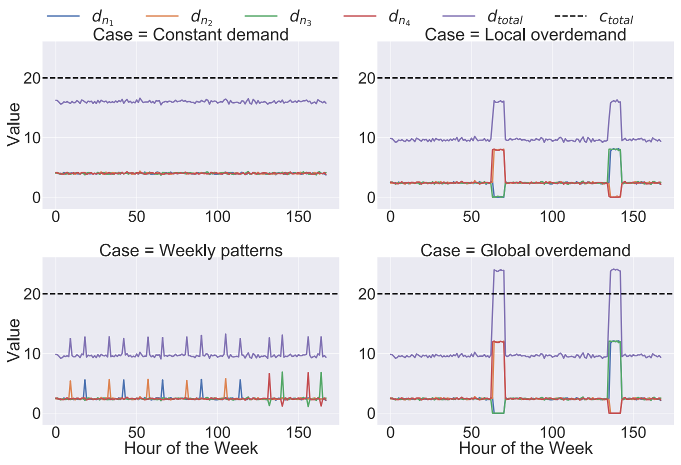

Fig. 7. Example of synthetic data generated for the four scenarios used for sensitivity analysis.

V. Experimental Setup

In this section, we first present the experimental setup used for evaluating our solution by means of synthetic and real-world data. We then describe the scenarios that are adopted for our sensitivity analysis. Finally, we discuss the different baselines that are used to evaluate the performance of PRORL.

A. Real-world Data

Dataset. We evaluate PRORL using a publicly available dataset composed of network traces gathered in the city of Milan for two months in 2013 [13]. Herein, the position of the base stations is hidden, while data is collected over square cells with a size of $235 \times 235\ m^2$. Inside each cell, traffic volume is aggregated and anonymized every ten minutes. A new trace is generated every time a user receives or sends SMS / calls or an Internet connection starts / ends. We then combine the position of the grid cells with the estimated position of real base stations obtained from a public dataset [43].

In our experiments, we consider 12 base stations that correspond to the PoPs set $\mathcal{N}$. Since the distribution of demands is highly skewed – most base stations have very low demands, while a few handle the majority of traffic – we discard the ones with the lowest peaks in the dataset, since they would represent an unrealistic PoP, where a single resource unit is always underutilized due to the low resource requirement. To consider a challenging scenario for resource allocation, we choose 6 unique pairs of base stations characterized by the highest distance in terms of demand profile over time. By doing so, we select nodes for which demand peaks at different times of the day or the week. Figure 5 presents the demand for the selected nodes over the 8 weeks of data.

Training and Evaluation Procedure. We train our solution by cycling through the data of the *training set* for a total of 336000 steps. The number of training steps is chosen empirically on the basis of validation performance, which improves rapidly in the early stages of training while tending to plateau as training progresses, as shown in Figure 6.[2] Every time the entire training set is seen by the agent or the attempts D are exhausted, an episode terminates. Every five episodes, a validation run is performed over an unseen set of data (the *validation set*), and the agent's model is saved if a new best score is obtained. At the end of training, the model that obtains the best score at validation time is evaluated over an additional set of unseen data, the *evaluation set*. We obtain the three datasets from the 8 weeks by temporally splitting the data: 6 weeks for the training set and one week for the validation and the evaluation sets respectively. The performance we report refers to the score obtained on the evaluation set. To ensure statistical validity of the results, we use 10 runs, each using a different random initialization of neural network parameters.

Agent Setup. We use a fully-connected network with ReLU activations and number of hidden layer units $\in \{\{64, 128, 64\}, \{64, 128, 256, 128\}\}$, trained using the Adam optimizer [44]. We consider learning rates $lr \in \{0.001, 0.005, 0.0001, 0.0005\}$ and batch sizes $batch\_size \in \{32, 64, 128, 256\}$. The exploration is based on an ϵ-greedy policy, with ϵ linearly decreased from 1 to 0.05 during the first half of the training and fixed to 0.05 for the remaining steps. The replay buffer has a capacity equal to 10000 and we start filling it with 9000 steps of bootstrapping in which we perform no learning and a random policy is used. The target network update frequency is 1000, and we use a discount factor $\gamma = 0.99$. From a grid search, we find that the best performance is achieved with the following set of values: $lr = 0.005$, $batch\_size = 256$ and $net = \{64, 128, 64\}$.

Environment Setup. Overall, we set the environment with 180 available resource units with size $\sigma = 890$, where the total number of units is obtained from the base station density we observe from the dataset, whereas σ is set in order to have a total demand at most equal to the 80% of the total capacity. We initialize the environment with 10 units for each PoP, while the remaining are set in the pool $\mathcal{B}$. We use $D = 150$ attempts, having explored values of $D \in \{150, 300, 600, 1000\}$.

We set $\tau = 0$ and the time step size t to 1 hour. The time step size influences the resolution at which the demand is collected and evaluated. The choice of the time interval is of key importance. If it is too short, the changes might be temporary and, in general, very noisy. If it is too long, the algorithm will not be able to adapt to non-negligible demand variations that might happen between the two sampling points. Figure 5 shows that, for the considered real-world dataset, this is an appropriate level of granularity to consider. It is worth noting that, given a scenario with different demand variability, our solution can support a significantly smaller granularity that approaches real time, as the average time required to execute one decision is less than 2 milliseconds. In general, the use of a learned model in such optimization problems may be appropriate when decisions need to be made rapidly and running a solver is unfeasible [42].

Finally, we train and evaluate PRORL using four different configurations of weights for our utility function (Equation 9):

- ***Efficiency***: we prioritize optimization of the R^{surplus} component by using the weights $\{w_{\text{surplus}} = 0.6, w_{\text{cost}} = 0.05, w_{\text{gap}} = 0.35\}$;
- ***Balanced***: We balance the trade-off between the objective components by using the following set of weights $\{w_{\text{surplus}} = 0.3, w_{\text{cost}} = 0.1, w_{\text{gap}} = 0.6\}$;
- ***Quality of Service***: we prioritize optimization of the demand satisfaction component R^{gap} by using the weights $\{w_{\text{surplus}} = 0.2, w_{\text{cost}} = 0.05, w_{\text{gap}} = 0.75\}$;
- ***Strict Quality of Service***: we optimize mainly R^{gap} while disregarding the cost, by using the weights $\{w_{\text{surplus}} = 0.1, w_{\text{cost}} = 0, w_{\text{gap}} = 0.9\}$.

We note that the weight associated with the cost w_{cost} is always smaller than the others since, if it is given too high importance, the behavior of the agent degenerates into always waiting (since it does not incur a cost).

B. Sensitivity Analysis

We train and evaluate PRORL considering four challenging cases in order to demonstrate the robustness of our solution. We use a synthetic data generator where the demand is drawn from a set of Gaussian distributions. The generator allows for setting different values of means and standard deviations

[2]We are also aware that more complex problem settings (e.g., more PoPs) would likely require more training steps. However, determining a *priori* how many steps are required to attain satisfactory performance is generally not possible in the RL with function approximation setting, requiring this type of empirical methodology that we have followed in our experimentation work.

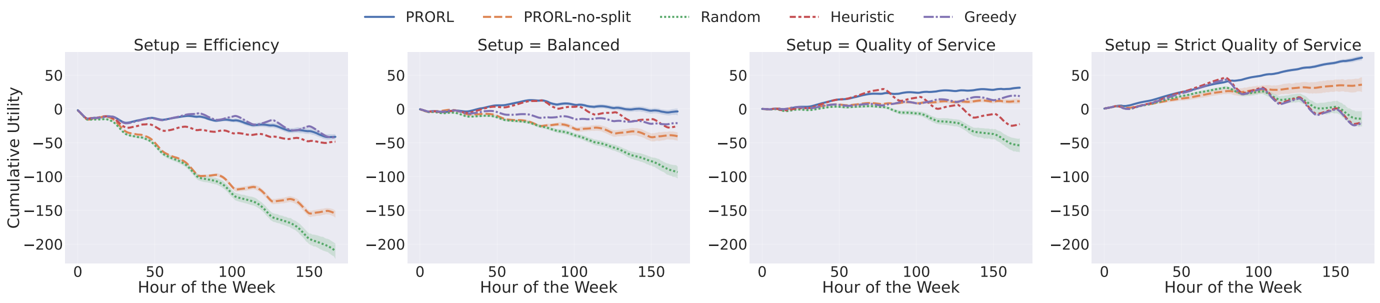

Fig. 8. Cumulative utility at evaluation time over the real-world dataset for the different utility setups (the higher the better).

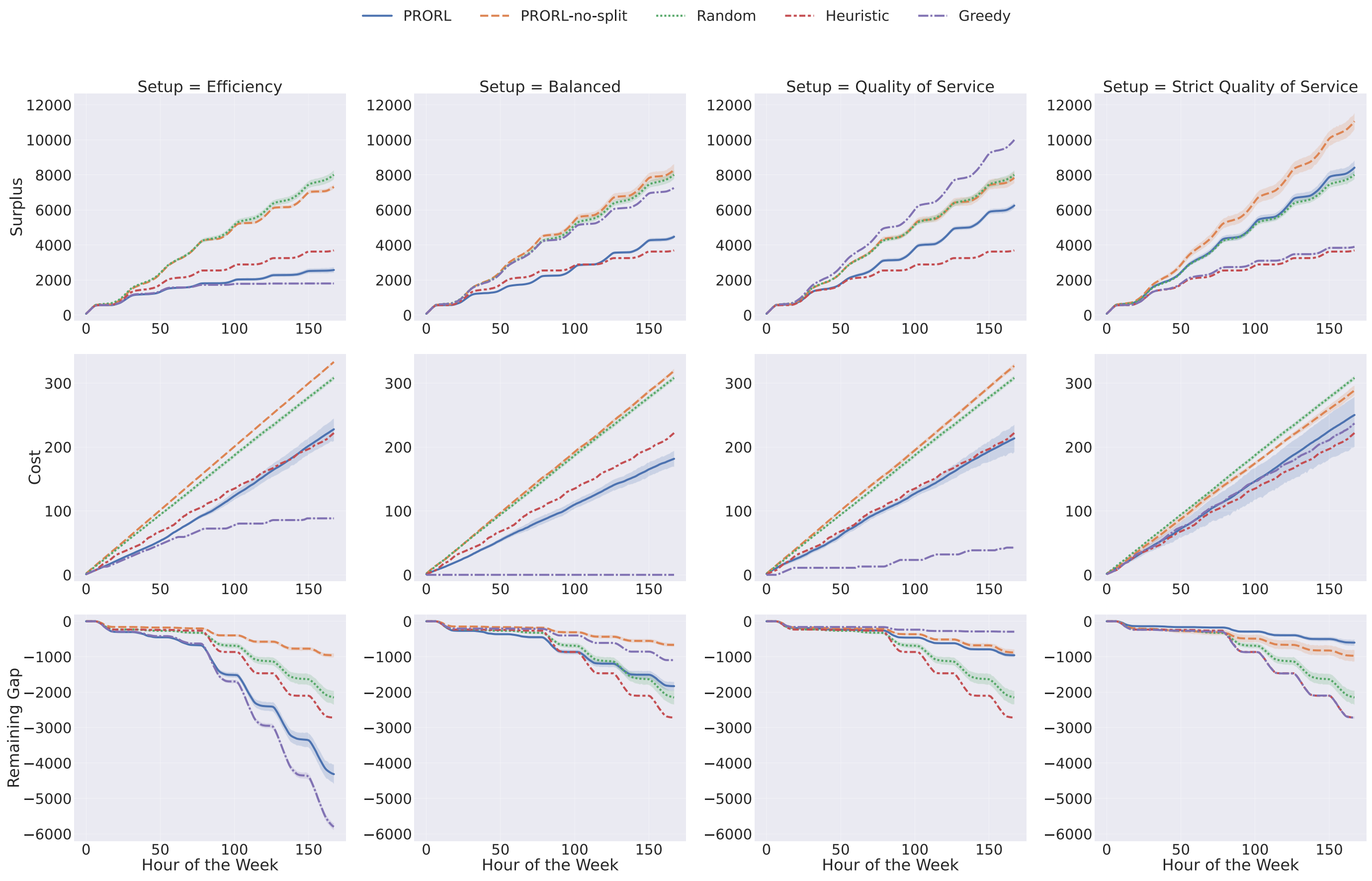

Fig. 9. Cumulative surplus, cost (for both the lower the better), and remaining gap (the higher the better) at evaluation time on the real-world dataset for the different utility setups. These results illustrate the ability of PRORL to differently prioritize components of the utility function, depending on their weights.

and to create peaks of demand with periodic or sporadic patterns over a given time interval. In particular, we create four different scenarios, as depicted in Figure 7:

1) ***Constant demand***: we keep the demand constant on average for the entire week, as shown in the top left part of Figure 7. It is a simple case, however, for which proactivity in the allocation does not give any gains.
2) ***Local overdemand***: we keep the demand constant for most of the week, but twice in a week peaks of demand occur for some hours at a given node, as shown in Figure 7 top right. This scenario represents a relevant test for a proactive agent that aims to learn when the peak will occur and prevent cases in which the demand is not satisfied or resources are wasted.
3) ***Weekly patterns***: we simulate a typical week in an urban area, in which a clear distinction between residential and business districts is present. During the weekdays, people commute from home to work during the morning and vice-versa in the evening, while on the weekend demand peaks occur in different areas (see Figure 7 bottom left).
4) ***Global overdemand***: we analyze a case in which the total demand surpasses the total capacity of the system (see Figure 7 bottom right). This is an extreme case, where complete demand satisfaction is not feasible. PRORL should be able to limit the movement actions and to reduce the associated management cost.

All the cases share the same environment configuration: we use a PoP set composed of 4 nodes and a pool of 20 resource

TABLE II
AVERAGE TOTAL REWARD AND CONFIDENCE INTERVAL FOR THE FOUR SCENARIOS USED FOR SENSITIVITY ANALYSIS USING THE UTILITY CONFIGURATION REFERRED TO AS "BALANCED".

		Utility (↑)	Surplus (↓)	Cost (↓)	Remaining Gap (↑)
Constant demand	Random	-53.163 ± 16.842	385.6 ± 184.4	248.7 ± 6.8	-1546.6 ± 193.9
	Heuristic	116.861 ± 1.062	155.0 ± 7.3	236.3 ± 4.7	-250.2 ± 6.8
	Greedy	116.675 ± 0.574	156.0 ± 6.6	234.5 ± 3.2	-252.0 ± 3.1
	PRORL-no-split	$\mathbf{137.995 \pm 0.157}$	310.8 ± 0.5	16.9 ± 0.6	-87.6 ± 0.9
	PRORL	$\mathbf{137.693 \pm 0.241}$	310.6 ± 0.9	20.3 ± 2.7	-88.8 ± 1.1
Local overdemand	Random	5.656 ± 14.373	1099.9 ± 252.5	248.8 ± 6.2	-699.7 ± 119.2
	Heuristic	140.784 ± 0.352	99.1 ± 1.4	55.8 ± 0.5	-154.0 ± 2.5
	Greedy	141.912 ± 0.589	89.6 ± 5.3	54.0 ± 1.2	-150.1 ± 2.8
	PRORL-no-split	121.477 ± 3.396	248.4 ± 49.9	331.9 ± 1.8	-125.7 ± 31.2
	PRORL	$\mathbf{153.359 \pm 0.3}$	99.2 ± 6.5	53.3 ± 2.7	-50.2 ± 3.0
Weekly patterns	Random	14.596 ± 12.932	1081.9 ± 249.9	248.8 ± 6.2	-634.2 ± 112.7
	Heuristic	159.12 ± 0.0	18.0 ± 0.0	36.0 ± 0.0	-50.0 ± 0.0
	Greedy	159.12 ± 0.0	18.0 ± 0.0	36.0 ± 0.0	-50.0 ± 0.0
	PRORL-no-split	137.547 ± 2.164	149.0 ± 25.4	334.5 ± 0.8	-40.4 ± 29.6
	PRORL	$\mathbf{166.984 \pm 0.006}$	1.0 ± 0.0	0.4 ± 0.6	-7.8 ± 0.3
Global overdemand	Random	-7.874 ± 13.628	1105.0 ± 249.5	248.8 ± 6.2	-809.9 ± 117.0
	Heuristic	126.248 ± 0.240	108.0 ± 0.0	56.0 ± 0.0	-270.6 ± 2.0
	Greedy	126.248 ± 0.240	108.0 ± 0.0	56.0 ± 0.0	-270.6 ± 2.0
	PRORL-no-split	108.688 ± 2.583	301.9 ± 66.6	323.8 ± 12.4	-208.9 ± 23.3
	PRORL	$\mathbf{139.213 \pm 0.134}$	122.8 ± 8.4	59.5 ± 2.9	-153.7 ± 4.8

units. We set the resource unit size $\sigma = 10$ and the utility weights to the values corresponding to the *Balanced* setup. The standard deviation of the synthetic generator is equal to 1. In this case, in addition to the agent's initial state, we also initialize the synthetic generator using 10 different random seeds to ensure statistical validity of the results. We explored the hyperparameters of our agent and we found that the best configuration used for the real-world dataset is the best also for the synthetic data. Due to the simpler nature of the problem, we set the experience replay capacity to 6000, the bootstrapping phase contains 3000 steps, and the total number of training steps was set to 86400 for all the cases apart from the *Weekly patterns* case that was trained for 168000 steps.

C. Baselines

We evaluate PRORL by comparing its performance against the following baselines:

- ***Random policy*** picks the two sub-actions by sampling uniformly from the action spaces $\mathcal{A}(S_t)$;
- ***Heuristic policy*** always selects the PoPs with the highest remaining gap and the highest surplus (if any) for the n_{add} and n_{remove} actions respectively, otherwise it waits;
- ***Greedy policy*** emulates the movement for each of the actions that are one time step away in the MDP, and greedily chooses the action resulting in the highest utility;
- ***PRORL-no-split*** is an RL agent identical to the proposed solution, with the sole exception that the action space has not been split into two sub-actions, resulting in a combinatorial set of possible actions.

VI. EXPERIMENTAL RESULTS

A. Evaluation using Real-world Data

Figure 8 presents the cumulative utility for the 4 different configurations of the utility function, while Figure 9 illustrates the cumulative surplus (first row), movement cost (second row), and remaining gap (last row). Overall, the results confirm the superiority of PRORL in simultaneously optimizing the 3 objectives. Only in the *Efficiency* setup the *Greedy* baseline obtains similar total utility, while in the remaining setups our solution greatly outperforms the four baselines. In terms of single objectives, let us highlight the diverse behaviors given the fact that the utility function setup weighs the 3 objective components differently. The different prioritization of the remaining gap component is evident between the *Efficiency* and the *Strict Quality of Service* setups. The opposite behavior occurs for the surplus: in the *Efficiency* setup the surplus is one of the lowest among the baselines, while the remaining gap is one of the highest. In contrast, a static policy such as the *Heuristic* cannot prioritize different components of the objective differently, and maintains identical values across the different setups. In terms of cost, the *Balanced* setup shows that PRORL can satisfy the demand while reducing the movement cost, while the *Greedy* baseline always waits since it leads to the highest reward in the short term.

Finally, analyzing the performance of the two PRORL agents, the advantage of splitting the action space is immediately apparent. In fact, the performance of PRORL-no-split is considerably worse than that of PRORL in all four setups. In particular, in the *Efficiency* and *Balanced* setups, PRORL-no-split can only outperform a *Random* policy, while in the others it can compete with other baselines. We believe that the reason for such poor performance is the greatly increased action space size, which indeed requires a much longer training phase to converge. On the other hand, PRORL is more efficient in requiring fewer steps to derive an effective policy, thus resulting in superior resource allocation decisions.

B. Sensitivity Analysis with Synthetic Data

Table II presents the overall results obtained using synthetic data for the 4 sensitivity analysis cases. The columns show the average total utility as well as the breakdown into the total surplus, total cost, and total remaining gap. Overall, the results show the superiority of PRORL when the data exhibits well-defined patterns - in fact, in the *Weekly patterns* scenario, our agent outperforms the baselines on all the three components of the objective. In the *Constant demand* scenario, our agent trades off a surplus of resources in order to achieve more than $3\times$ better remaining gap and more than $10\times$ better cost. Instead, the baselines perform continuous movements due to the high variation of demand generated in this scenario, resulting in a larger surplus but much worse remaining gap and cost. In the *Local overdemand* scenario, our agent results in the cheapest solution that satisfies the nodes' demands the most (i.e., it has the highest total remaining gap). Also in this setting, it trades off a surplus of resources for higher total utility. Considering a more challenging scenario, such as the *Global overdemand*, PRORL is still able to outperform the baseline in terms of total utility and remaining gap by trading off performance in the other components of the objective.

Finally, comparing the results for our two agents, the importance of reducing the action space size is once again evident, even in the case in which the number of possible actions is small (as we only have 4 nodes). In fact, the performance of PRORL-no-split is always lower than that of its optimized counterpart, PRORL, as well as the *Heuristic* and *Greedy* baselines. The only exception is the *Constant demand* scenario, in which the performance results are very similar. This is motivated by the fact that, in this scenario, a good solution requires waiting most of the time since the load is constant; therefore, in small setups, having a single agent to train and no coordination among the two sub-agents might allow the discovery of similar resource allocation policies.

VII. Conclusions

In this work, we have demonstrated the advantages of Deep Reinforcement Learning as an approach for orchestrating resources in O-RAN deployments. Our solution is capable of learning the dynamics of a complex environment in which it operates and make decisions that lead to outperforming greedy solutions - both over real and synthetic data - while optimizing three competing components of the objective, namely: demand satisfaction, resource utilization, and the cost associated to resource movements. Moreover, we have demonstrated the flexibility of PRORL, showing that it can be used to tune the optimization of competing objectives (and corresponding trade-offs) by changing their weights. Finally, we have also proven the robustness of our method in challenging scenarios characterized by high variability in terms of demand profiles through an extensive sensitivity analysis.

Our future research agenda will focus on the deployment of PRORL in a real O-RAN deployment environment, which handles users' traffic and demands, and on its extension for orchestration of the lower levels of the O-RAN stack.

References

[1] T. L. Marzetta, "Noncooperative Cellular Wireless with Unlimited Numbers of Base Station Antennas," *IEEE Transactions on Wireless Communications*, vol. 9, no. 11, pp. 3590–3600, 2010.

[2] I. F. Akyildiz, J. M. Jornet, and C. Han, "Terahertz band: Next frontier for wireless communications," *Physical Communication*, vol. 12, pp. 16–32, 2014.

[3] A. Bourdoux, A. N. Barreto, B. van Liempd, C. de Lima, D. Dardari, D. Belot, E.-S. Lohan, G. Seco-Granados, H. Sarieddeen, H. Wymeersch, J. Suutala, J. Saloranta, M. Guillaud, M. Isomursu, M. Valkama, M. R. K. Aziz, R. Berkvens, T. Sanguanpuak, T. Svensson, and Y. Miao, "6G White Paper on Localization and Sensing," 2020.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Computer Communication Review*, vol. 38, no. 2, p. 69–74, 2008.

[5] T. O'Shea and J. Hoydis, "An Introduction to Deep Learning for the Physical Layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, 2017.

[6] O-RAN Alliance. (2019) O-RAN Software Community. [Online]. Available: https://www.o-ran.org/

[7] 3GPP, "Study on New Radio Access Technology: Radio Access Architecture and Interfaces," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 38.801, 2017, version 14.0.0. [Online]. Available: http://www.3gpp.org/DynaReport/38801.htm

[8] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges," *IEEE Communications Surveys & Tutorials*, pp. 1–1, 2023.

[9] O-RAN Working Group 1, "O-RAN Operations and Maintenance Interface 4.0," Technical Specification (TS), 2020, O-RAN.WG1.O1-Interface.0-v04.00.

[10] ——, "NR; Radio Resource Control (RRC); Protocol Specification," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.331, 2018, version 15.0.0. [Online]. Available: http://www.3gpp.org/DynaReport/38331.htm

[11] ——, "Evolved Universal Terrestrial Radio Access (E-UTRA) and NR; Service Data Adaptation Protocol (SDAP) Specification," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 37.324, 2022, version 17.0.0. [Online]. Available: http:////www.3gpp.org/DynaReport/37324.htm

[12] M. Shehata, A. Elbanna, F. Musumeci, and M. Tornatore, "Multiplexing Gain and Processing Savings of 5G Radio-Access-Network Functional Splits," *IEEE Transactions on Green Communications and Networking*, vol. 2, no. 4, pp. 982–991, 2018.

[13] G. Barlacchi, M. De Nadai, R. Larcher, A. Casella, C. Chitic, G. Torrisi, F. Antonelli, A. Vespignani, A. Pentland, and B. Lepri, "A multi-source dataset of urban life in the city of Milan and the Province of Trentino," *Scientific Data*, vol. 2, no. 1, p. 150055, 2015.

[14] L. Chen, T.-M.-T. Nguyen, D. Yang, M. Nogueira, C. Wang, and D. Zhang, "Data-Driven C-RAN Optimization Exploiting Traffic and Mobility Dynamics of Mobile Users," *IEEE Transactions on Mobile Computing*, vol. 20, no. 5, pp. 1773–1788, 2021.

[15] T. Pamuklu, M. Erol-Kantarci, and C. Ersoy, "Reinforcement Learning Based Dynamic Function Splitting in Disaggregated Green Open RANs," in *IEEE ICC'21*, 2021, pp. 1–6.

[16] E. Coronado, S. Siddiqui, and R. Riggio, "Roadrunner: O-RAN-based Cell Selection in Beyond 5G Networks," in *NOMS'22*, 2022, pp. 1–7.

[17] R. Li, Z. Zhao, Q. Sun, C.-L. I, C. Yang, X. Chen, M. Zhao, and H. Zhang, "Deep Reinforcement Learning for Resource Management in Network Slicing," *IEEE Access*, vol. 6, pp. 74 429–74 441, 2018.

[18] M. Yan, G. Feng, J. Zhou, Y. Sun, and Y.-C. Liang, "Intelligent Resource Scheduling for 5G Radio Access Network Slicing," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 7691–7703, 2019.

[19] M. Peng, K. Zhang, J. Jiang, J. Wang, and W. Wang, "Energy-Efficient Resource Assignment and Power Allocation in Heterogeneous Cloud Radio Access Networks," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 11, pp. 5275–5287, 2015.

[20] M. Peng, Y. Wang, T. Dang, and Z. Yan, "Cost-Efficient Resource Allocation in Cloud Radio Access Networks With Heterogeneous Fronthaul Expenditures," *IEEE Transactions on Wireless Communications*, vol. 16, no. 7, pp. 4626–4638, 2017.

[21] X. Wang, J. D. Thomas, R. J. Piechocki, S. Kapoor, R. Santos-Rodríguez, and A. Parekh, "Self-play learning strategies for resource assignment in Open-RAN networks," *Computer Networks*, vol. 206, p. 108682, 2022.

[22] F. Mungari, "An RL Approach for Radio Resource Management in the O-RAN Architecture," in *SECON'21*, 2021, pp. 1–2.

[23] S.-Y. Lien, D.-J. Deng, and B.-C. Chang, "Session Management for URLLC in 5G Open Radio Access Network: A Machine Learning Approach," in *IWCMC'12*, 2021, pp. 2050–2055.

[24] L. Chen, D. Yang, D. Zhang, C. Wang, J. Li, and T.-M.-T. Nguyen, "Deep mobile traffic forecast and complementary base station clustering for C-RAN optimization," *Journal of Network and*

Computer Applications, vol. 121, pp. 59–69, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804518302455
[25] G. Sun, Z. T. Gebrekidan, G. O. Boateng, D. Ayepah-Mensah, and W. Jiang, "Dynamic Reservation and Deep Reinforcement Learning Based Autonomous Resource Slicing for Virtualized Radio Access Networks," *IEEE Access*, vol. 7, pp. 45 758–45 772, 2019.
[26] S. Mondal and M. Ruffini, "Optical Front/Mid-haul with Open Access-Edge Server Deployment Framework for Sliced O-RAN," *IEEE Transactions on Network and Service Management*, 2022.
[27] E. Sarikaya and E. Onur, "Placement of 5G RAN Slices in Multi-tier O-RAN 5G Networks with Flexible Functional Splits," in *CNSM'21*, 2021, pp. 274–282.
[28] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "ColO-RAN: Developing Machine Learning-based xApps for Open RAN Closed-loop Control on Programmable Experimental Platforms," *IEEE Transactions on Mobile Computing*, pp. 1–14, 2022.
[29] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[30] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley, "A survey of multi-objective sequential decision-making," *Journal of Artificial Intelligence Research*, vol. 48, pp. 67–113, 2013.
[31] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari With Deep Reinforcement Learning," in *NeurIPS'13*, 2013.
[32] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[33] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3, pp. 293–321, 1992.
[34] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *AAAI'18*, 2018.
[35] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," in *AAAI'16*, 2016.
[36] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," in *ICLR'16*, 2016.
[37] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *ICML'16*, 2016.
[38] E. A. Silver, "Operations research in inventory management: A review and critique," *Operations Research*, vol. 29, no. 4, pp. 628–645, 1981.
[39] M. Florian, J. K. Lenstra, and A. Rinnooy Kan, "Deterministic production planning: Algorithms and complexity," *Management Science*, vol. 26, no. 7, pp. 669–679, 1980.
[40] A. J. Wood, B. F. Wollenberg, and G. B. Sheblé, *Power Generation, Operation, and Control*. John Wiley & Sons, 2013.
[41] R. Baldick, "The generalized unit commitment problem," *IEEE Transactions on Power Systems*, vol. 10, no. 1, pp. 465–475, 1995.
[42] Y. Bengio, A. Lodi, and A. Prouvost, "Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon," *European Journal of Operational Research*, vol. 290, pp. 405–421, 2021.
[43] Unwired Labs. (2008) OpenCelliD - Open Database of Cell Towers & Geolocation. [Online]. Available: https://opencellid.org/
[44] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *ICLR'15*, 2015.

Alessandro Staffolani is a Ph.D. candidate in Computer Science and Engineering at University of Bologna, Italy. He is interested in learning-based approaches for addressing the optimization of resources by means of scheduling and orchestration, in the context of distributed systems and network infrastructure.

Victor-Alexandru Darvariu is a Postdoctoral Research Fellow in the Department of Computer Science at University College London, UK, where he also recently received his PhD. He is interested in tackling combinatorial optimization problems with learning-based approaches spanning reinforcement learning, planning and graph neural networks, as well as their applications in infrastructure networks and systems.

Luca Foschini received MSc and PhD degrees from the University of Bologna, Italy, where he is Associate Professor of Computer Engineering. His interests include distributed systems and solutions for system and service management, management of cloud computing, context data distribution smart city scenarios, and management of O-RAN and 5G virtualized telco infrastructures. He has published around 240 papers, with around 90 of them on the major international journals. He is part of the IEEE ComSoc BoG where he serves as EMEA Director.

Michele Girolami received his M.Sc. and Ph.D. in Computer Science from the University of Pisa in 2007 and 2015, respectively. Currently he is a Researcher at ISTI-CNR at the Wireless Network Laboratory. He participates to several EU and national projects. His research interests are mainly focused on indoor localization, proximity detection, pervasive computing and Internet of Things. He has been involved in the organization of several international workshops and conferences.

Paolo Bellavista received MSc and PhD degrees in computer science engineering from the University of Bologna, Italy, where he is a full professor of distributed and mobile systems. His research activities span from pervasive wireless computing to online big data processing under quality constraints, from edge cloud computing to middleware for Industry 4.0 applications. He has published around 300 papers, with around 120 of them in major international journals in the above fields. He serves on several Editorial Boards of leading IEEE and ACM journals.

Mirco Musolesi is Full Professor of Computer Science at the Department of Computer Science at University College London. He is also Full Professor of Computer Science at the Department of Computer Science and Engineering at the University of Bologna. Previously, he held research and teaching positions at Dartmouth, Cambridge, St Andrews and Birmingham. The focus of his lab is on Artificial Intelligence and its applications to a variety of practical and theoretical problems and domains.