Fast Context Adaptation in Cost-Aware Continual Learning

Seyyidahmed Lahmer, *Student Member, IEEE,* Federico Mason, *Member, IEEE,* Federico Chiariotti, *Member, IEEE,* and Andrea Zanella, *Senior Member, IEEE*

Abstract—In the past few years, DRL has become a valuable solution to automatically learn efficient resource management strategies in complex networks with time-varying statistics. However, the increased complexity of 5G and Beyond networks requires correspondingly more complex learning agents and the learning process itself might end up competing with users for communication and computational resources. This creates friction: on the one hand, the learning process needs resources to quickly convergence to an *effective* strategy; on the other hand, the learning process needs to be *efficient*, i.e., take as few resources as possible from the user's data plane, so as not to throttle users' QoS.
In this paper, we investigate this trade-off and propose a dynamic strategy to balance the resources assigned to the data plane and those reserved for learning. With the proposed approach, a learning agent can quickly converge to an efficient resource allocation strategy and adapt to changes in the environment as for the CL paradigm, while minimizing the impact on the users' QoS. Simulation results show that the proposed method outperforms static allocation methods with minimal learning overhead, almost reaching the performance of an ideal out-of-band CL solution.

Index Terms—Resource allocation, Reinforcement learning, Cost of learning, Continual learning, Meta-learning, Mobile Edge Computing.

1 Introduction

The role of Artificial Intelligence (AI) in communication networks has become more and more central with the transition from 4G to 5G, and learning is at the core of the 6G standardization process [1]. Mobile networks are no longer designed as rigid entities that the final users have to adapt to, rather are becoming customizable services evolving according to the users' needs [2]. The Network Slicing (NS) paradigm supports this approach by enabling the definition of multiple logical networks overlaying the same physical infrastructure [3], with each *slice* devoted to a specific class of service. This allows applications with very different requirements to coexist and share spectrum resources. However, managing NS, as well as other advanced application scenarios, requires judicious allocation of both transmission and computational resources to users, according to their Quality of Service (QoS) targets, in a fast-paced scenario [4], which is expected to become even more straining with 6G.

Hand-designed resource allocation strategies may not be up to this challenge, so that growing attention has been dedicated to machine-learning approaches. In particular, Deep Reinforcement Learning (DRL) is considered a promising framework for deriving adaptable and robust strategies for network orchestration [4] and resource allocation [5].

DRL's *effectiveness* in dealing with complex scenarios is indeed well-established: with proper training, the DRL agents can find foresighted policies aiming for long-term objectives [6], significantly improving network performance. Such promising results, however, have been typically obtained in *stationary* environments: if this assumption is not satisfied, the performance of pre-trained DRL agents may dramatically decrease when the network dynamic shifts away from the training environment.

Approaches based on the Continual Learning (CL) paradigm [7] are designed to deal with non-stationary systems. CL enables the adaptation of a learning agent to a series of subsequent tasks that, in a network scenario, may represent different network configurations. We observe that combining CL and DRL for managing network resources in non-stationary scenarios has a non-negligible cost in terms of energy, computation, and communication resources [8]. These resources are necessarily subtracted to the *data plane*, i.e., the part of the system that is responsible for transmitting, processing, and forwarding user data packets. Therefore, supporting the learning represents an overhead for the system, which can negatively impact users' QoS. We use the term *cost of learning* to indicate the impact that the learning process can have on users' performance due to the resources it requires. Considering such a cost implies that the learning framework, in addition to being effective, must also be *efficient*, that is, require as few resources as possible to achieve its goals. The cost of learning problem is particularly critical considering the ever-larger size of most recent DRL neural networks, and the growing demand for efficient systems, as for the green networking paradigm [9]. We observe that Mobile Edge Computing (MEC) [10] solutions does not solve the problem, but just move it at the network edge. In fact, while MEC allows computationally-expensive tasks (such as the training of DRL algorithms) to be carried

This work was supported by the EU H2020 MSCA ITN project Greenedge (grant no. 953775) and by the European Union, under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, as part of the REDIAL (SoE0000009) Young Researchers grant and the partnership on "Telecommunications of the Future" (PE0000001) - program "RESTART."

out directly in dedicated edge nodes physically close to the data sources, still the limited transmission, computational and energetic resources of such nodes have to be shared between data and learning planes.

Finding a balance between the number of resources to be used for improving the system reactivity to variations and those to be allocated for serving the users may be a very difficult task. This is particularly critical in the case of CL systems, in which agents must constantly adapt to new working conditions. As the very same network resources are also used for the training, a trade-off between the capability of DRL agent to learn new tasks and its performance during the current task arises.

Note that, although at a first sight this problem may recall the well-known exploration-exploitation problem in learning systems, it is actually fundamentally different. In fact, the exploration-exploitation problem involves finding a balance between exploring new strategies, with the risk of temporarily losing some performance, and exploiting the currently learned strategy that, however, may be globally suboptimal, thus wasting part of the system capacity. In this setting, the resources required by the learning process are typically ignored: whatever action is taken, the outcome is assumed to be available to the learner, enriching its experience, without any cost. In this paper, instead, we look at the resources needed to transfer such information to the learner and turn it into experience, irrespective of whether it originates from an exploration or exploitation action. From a theoretical perspective, the scenario we look at is hence a Meta Learning (MeL) one, in which the agent's actions determine the efficiency of the learning data aggregation and processing.

The cost of learning is therefore a fundamental aspect to be considered in modern network design, and recent works have proposed learning-based frameworks that are computation-aware [11]. Despite the high interest of the scientific community in this field, the cost of learning for DRL models is still a relatively unexplored subject in the networking literature, and even the most recent works on resource allocation and NS ignore the true cost of combining DRL and CL in modern networks [12], making the effectiveness of state of the art DRL solutions questionable.

In this work, we analyze the trade-off between *effectiveness* and *efficiency* in CL, formally defining the resource allocation problem and presenting a heuristic solution to allocate resources to the data and learning planes. The proposed scheme effectively controls training in a CL framework, maximizing the efficiency of the training (i.e., reducing the learning plane overhead) while still achieving effective resource allocation (i.e., the same QoS as the ideal approach that assumes learning does not consume users' resources) in a reasonable time. Although we applied our optimization framework to a networking scenario, it is actually more general and can be adapted to any learning-based allocation problem in which the allocated resources are also required for the agent training, such as MEC job scheduling.

The major contributions of our work are the following:

- We define a theoretical model to analyze the trade-off between *effectiveness* and *efficiency* of learning-based resource allocation schemes;
- We propose a CL strategy to enable the resource allocation scheme to adapt to sudden changes in traffic dynamics;
- We test the proposed model in a NS use case, in which learning agent and system users compete the same network resources;
- We compare the benefits and drawbacks of the proposed approach against a static resource-sharing scheme between data and learning planes, and an ideal strategy that considers out-of-band resources for the agent training (or, equivalently, assumes the learning agent does not consume any user-plane resources). Our simulation results show that the proposed heuristic performs closely to the ideal (out-of-band) approach, minimizing the impact of learning plane traffic during the training.

A partial and preliminary version of this work was presented in [13]. In this paper, we extend that work by introducing the CL approach, providing a much richer set of results, and deepening the discussion and analysis of our observations.

The rest of this paper is organized as follows: first, Sec. 2 reports the most significant related work. We then present the model for optimizing data and learning plane in Sec. 3 and the NS use case definition in Sec. 4. Successively, Sec. 5 presents the simulation results. Finally, Sec. 6 concludes the paper and discusses some possible avenues for future work.

2 Related Work

While the latest advances in AI have made it possible to reach stunning performance levels in multiple fields, there is still a large gap between human cognition and AI models in terms of adaptation. Most of the current learning models need to be retrained from scratch every time a new task has to be accomplished, with a high cost in terms of computational power and time. For this purpose, the scientific community has recently leveraged the CL paradigm, which focuses on learning a series of subsequent data, associated with different tasks, without *catastrophically forgetting* the past knowledge [14]. Therefore, in CL scenarios the goal is to adapt to a time-varying environment, working on one task at a time and assuming that future information is inaccessible. This model appeals to the resource allocation problem considered in this manuscript, since in realistic networks the type, number and requirements of the users keep changing over time, making the system non-stationary (though stationarity can be assumed during the *coherence intervals*, i.e., the time periods during which the main system parameters do not change).

A baseline CL solution may involve a pre-trained model that is iteratively adapted to new tasks (or to changes in the environment), e.g., taking advantage of curriculum learning, as done in [15]. Replay-based methods form a more recent class of CL algorithms, which store past experience in memory or exploit a generative model to reproduce it, using this information as model input while training on new tasks [16], [17]. Regularization-based methods, which introduce a penalty term in the model's loss function with the goal of avoiding performance degradation in past tasks [18], [19], form another class of solutions. An extension of the

aforementioned class is proposed in [20], where the authors estimate the importance of each learned parameter and prevent the modifications of such parameters that most affect performance in past tasks. Finally, architecture-based methods define an additional branch of the model for each task, freezing the previously learned parameters when training the model on new scenarios [21], [22]. An example is provided in [23], where the authors developed a model organized into two blocks: the first is retrained every time a new task arises, while the latter distills the knowledge acquired for future reuse.

From a different perspective, CL algorithms aim at defining a strategy to detect the best settings for training a learning model in new scenarios. This concept falls within the MeL paradigm, which focuses on convergence speed and stability [24]. MeL methods differ in the output of the learning optimization, which may include the weight initialization strategy [25], the optimizer algorithm [26], the loss function [27], the dimensions of the learning architecture [28], and other hyper-parameters.

The methodology used for the MeL optimization task may be based on gradient descent [29], evolutionary algorithms [30], or learning-based approaches. For instance, the authors of [31] develop a CL model in which a DRL agent has to define the optimal settings of the task-specific block, balancing between validation accuracy and model complexity. Besides, MeL methods differ in terms of optimization goals, which may either be fully based on the model performance on a validation set or consider more specific aspects, such as the adaptability of the solution to multiple tasks [32], the greater importance of fast adaptation than asymptotic performance [33], and the difference between online and offline learning scenarios [34].

In particular, the use of MeL methods for optimizing DRL models is a relatively new field. In this scenario, the combination of CL and MeL avoids the need for a centralized agent that can handle each possible state-action pair and enables the definition of multiple and more straightforward policies. The authors of [35] show the benefits of MeL in a real-world scenario, analyzing a DRL robotic system that exploits a recurrent module to preserve past knowledge and speed up the training process. Interestingly, when applying MeL combined with DRL, a fundamental parameter is the choice of the exploration policy by which the agent interacts with the environment, which is an absent aspect in classification tasks. For instance, the authors of [36] develop a MeL model where prior information is used to define agnostic exploration policies enabling a better agent adaptation to multiple learning problems.

In the context of CL and MeL for 5G and 6G network management, drift detection is a critical task: if the environment changes abruptly, the MeL paradigm requires the learner to first become aware of the change, and delayed detection may lead to a violation of the service requirements [37]. Drifts can be classified as abrupt or gradual depending on the period during which the system performance degrades. The literature presents several drift detection algorithms, usually considering the prediction error of the learning model for estimating environment changes. It is possible to consider both heuristic [38] or learning-based strategies [39], with different advantages and drawbacks in terms of, e.g., false alarm probability and assumptions on the drift characteristics.

Despite the numerous works investigating CL and MeL in supervised and reinforcement learning scenarios, to the best of our knowledge none of them considers the trade-off between efficiency and effectiveness proposed in this manuscript. In the following sections, we will take advantage of solutions inspired by the literature, analyzing their impact in terms of both training efficiency and user performance. We will consider a baseline CL system where the models trained for previous tasks are stored in a central memory, similarly to what is done in [15]. Besides, we will consider a simple drift detection algorithm to monitor the environment statistics and trigger the retraining of the learning model. We chose relatively simple techniques for both drift detection and CL in order to focus on the main aspect of our analysis, which is the trade-off between efficiency and effectiveness in resource allocation problems, as represented by the cost of learning.

3 System Model

Let us consider a generic resource allocation problem, which is modeled as an infinite horizon Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathbf{P}, R, \gamma)$: $\mathcal{S}$ represents the state space, $\mathcal{A}$ is the action space (which is potentially different for each state), $\mathbf{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability matrix, which depends on the current state, the action chosen by the agent, and the landing state, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. Time is divided in slots, and the slot index is denoted by $t \in \mathbb{Z}^+$. The ultimate objective of a DRL agent is to find the optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$, which maximizes the expected long-term reward:

$$\pi^* = \underset{\pi:\mathcal{S}\to\mathcal{A}}{\arg\max}\, \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(\mathbf{s}_t, \pi(\mathbf{s}_t), \mathbf{s}_{t+1})\right]. \tag{1}$$

Let us assume that, in each time slot t, the system can allocate N resource blocks, which may represent communication bandwidth, computational cycles, or energy units, depending on the specific application: the type of resource may affect the definition of the specific MDP, but is immaterial for our reasoning. In the following we hence generally refer to a *request*, which can be a packet to be transmitted, a computing job to be executed, or an action to be taken, and we assume that each request requires exactly one resource block of some kind (transmission capacity, computational power, or energy).

The system resources are assumed to be partitioned into M different *slices*, where a slice may serve a single user, or a group of users with the same features. The action space then contains all possible resource allocation vectors that split the N resources among the M slices:

$$\mathcal{A} = \left\{ \mathbf{a} \in \{0, \dots, N\}^M : \sum_{m=1}^{M} a_m = N \right\}. \tag{2}$$

Furthermore, we assume that each slice is associated to a First-In First-Out (FIFO) queue of requests: each queue has a limited size Q, after which the system starts dropping older requests for that slice to make room for newer ones.

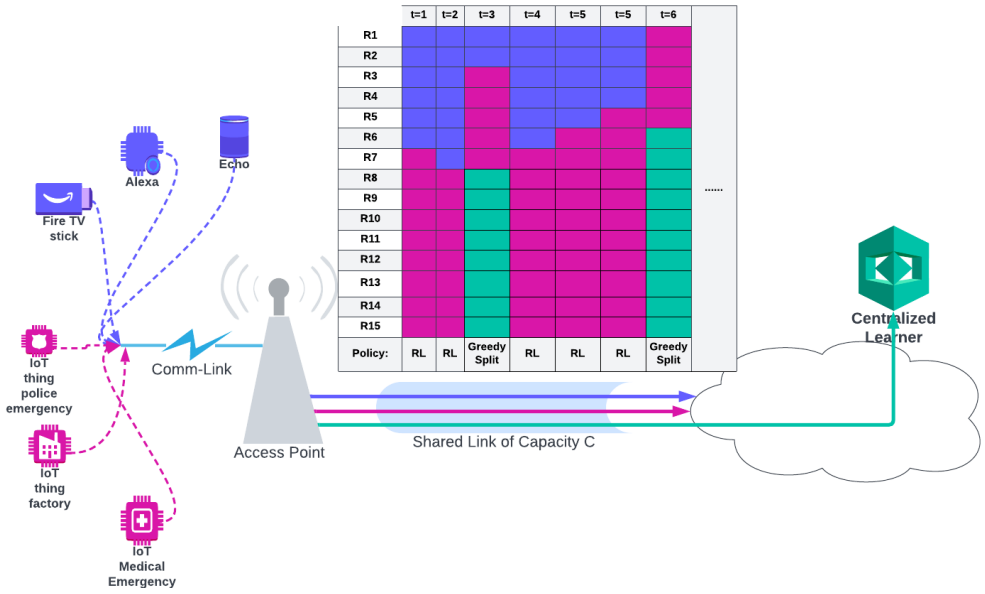


Fig. 1: Schematic of the learning control loop in a communication scenario: nodes on the left-hand side represent users, belonging to two different slices (blue and magenta). The connection between the base station and the Internet is used to carry both the users' traffic (blue and magenta streams) and the data needed to train the learner in the cloud (green stream), according to the allocation scheme graphically shown above the link.

In this work, we focus on Key Performance Indicators (KPIs) tied to the latency with which the requests of the different slices are served. However, the approach can be generalized to consider other metrics.

We hence indicate by $T_{m,i}$ the latency of the i-th request from slice m, which depends on the time it spends in the queue before being assigned a resource. Dropped or rejected requests have an infinite latency by definition. The i-th request from slice m is generated at time $t_{m,i}$, and age $\Delta_{m,i}(t)$ is defined as:

$$\Delta_{m,i}(t) = t - t_{m,i}. \tag{3}$$

We can then define the reward function:

$$R(\mathbf{s}, \mathbf{a}, \mathbf{s}') = \sum_{m=1}^{M} \sum_{i=1}^{a_m} f_m \left(\Delta_{q_m(i)} \right), \tag{4}$$

where $\Delta_{q_m(i)}$ is the age of the packet in position i of the m-th queue at the current time t, and $f_m : \mathbb{N} \to [0, 1]$ is a function mapping the latency of each request to slice m's resulting QoS. With a slight abuse of notation, we define $f(\varnothing) = 0$, where $\varnothing$ indicates that there is no packet in that position in the queue. We can distinguish between slices with *hard* timing requirements, for which the QoS of a request is 1 if it is served within a maximum latency, and 0 if it exceeds that deadline; and *soft* timing requirements, for which the QoS is a generic monotonically decreasing function of the latency.

We can then distinguish between *rejected* and *dropped* packets, the first being packets that find a full queue and are immediately discarded, the second referring to queued packets whose age is higher than the deadline and, in case of hard timing requirements, are hence dropped before service since they would not contribute to the QoS of the slice and just waste resources. We remark that only slices with hard timing requirements can experience dropped packets, while packet rejection can occur in any slice.

It should also be noted that dropped or rejected requests do not generate any rewards, as they are never included in the sum. The state of the system is then represented by the age of each request contained in each queue, so that in the most general case, $\mathcal{S} = (\{\varnothing\} \cup \mathbb{N})^{M \times Q}$.

The objective of the learning agent is then to learn how to allocate resources among users, so as to maximize their QoS parameters; it should also be aware of the slices that have a higher risk of violating hard timing requirements and schedule resources to avoid missing deadlines. However, learning is also a computational process, and the DRL agent may take up some of the same resources that may be allocated to the users in order to improve its policy. As we highlighted in our previous work [11], considering the cost of learning can lead to significantly different choices, limiting the amount and type of experience samples that are selected for training: this is also true regardless of the type of resource the learning requires.

However, even that work only considered static policies, which set up a separate virtual channel (either divided in time or in frequency) for the learning data, strictly separating the learning and data planes. Equivalently, an agent learning how to schedule tasks in an edge server could reserve a certain percentage of computation time to self-improvement, but the amount was decided in advance. This is clearly suboptimal: intuitively, the relative returns from policy self-improvement decrease over time, as the agent gradually converges to the optimal policy. After convergence, and as long as the environment statistics are stable, the value of further improvements to the policy is zero by definition. A dynamic policy for adapting the allocation between requests and learning should then take this into account.

Furthermore, the current state of the system also needs to be taken into account: if delaying the queued requests further does not have a large impact on the QoS, the

system can take away resources from the slices in order to improve the resource allocation policy, but if the impact is big, e.g., if some requests from a slice with hard timing requirements are already close to the deadline, they need to be prioritized, choosing immediate gains over potential future improvements.

This is particularly important for non-stationary environments, in which the coherence time of the MDP statistics is finite: in this kind of system, the learning agent needs to adapt the allocation to the changing statistics of the environment, and cannot rely on offline training, but must keep learning from experience and adapt to the changes proactively.

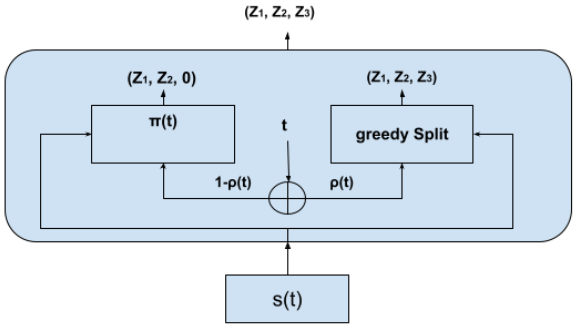


Fig. 2: Schematic of the learning plane resource allocation policy.

3.1 Learning Plane Resource Allocation

One of the problems of including the learning plane in the resource allocation policy is the circularity of the policy: in order to learn when to allocate resources to policy improvement, a DRL agent needs to first learn when learning is important. As the policy evolves over time and learning becomes less of a priority, this makes the reward that the agent perceives dependent on the agent's own reduced resources demand, making the learning more difficult.

In order to avoid this problem, we set an external rule to regulate learning, so that the environment that the agent sees is stationary. We define a generic resource allocation vector space $\mathcal{Z}$ as follows:

$$\mathcal{Z} = \left\{ \mathbf{z} \in \{0, \dots, N\}^M : \sum_{m=1}^{M} z_m \leq N \right\}. \tag{5}$$

We remark here that $\mathcal{Z}$ is not the action space, but rather a superset of it, i.e., $\mathcal{A} \subseteq \mathcal{Z}$: the definition of the action space in (2) indeed only considers allocations that assign all of the available resources to the users' slices, while $\mathcal{Z}$ also includes actions that allocate only part of the resources to the users: if $\sum_{m=1}^{M} z_m < N$, the remaining resources are allocated to the learning.

We can then divide the time slots in two categories, which we name *DRL* and *learning*: in DRL slots, all the resources are allocated to the users' slices according to the action chosen by the DRL agent, while in learning slots the resources are divided between the learning process and the users' slices, according to a simple, empirical strategy. We also remark that learning slots are not considered as experience samples for the DRL training, as the action $\mathbf{z}$ in these slots might not belong to the action space $\mathcal{A}$ considered in the DRL slots.

Fig. 1 shows a basic schematic of the process in the communication use case: the two classes of users, corresponding to Internet of Things (IoT) (magenta) and human communications (blue), transmit over a shared link, and the resources in each time slot (which correspond to bandwidth and time resources in the uplink to the Cloud) are allocated following a dynamic division. Slots 3 and 6 in the figure are learning slots: a significant portion of the resources is allocated to the learning plane (green line). We remind the reader that this networking example, while being the main motivation for our study, is not the only application of the model, which may also be used to allocate scarce computational or energy resources to different users in a dynamic scenario.

For the sake of simplicity, we assume that each slot will be used as a learning slot with probability $\rho(t)$, which decreases linearly over time as the learned policy becomes more stable. The actual shape of $\rho(t)$ (i.e., the learning curve) shall be defined based on the coherence time of the scenario, i.e., the number of slots τ over which the statistics of the environment will be approximately stationary. As explained later (see (14)) here we choose a linear function, but other choices are possible. We now need to define an allocation strategy in learning slots.

3.2 Greedy Allocation Strategy

To define an strategy for splitting resources in the learning slots, we consider two contrasting objectives: minimizing the loss of QoS for users, and maximizing the number of experience samples that can be transferred to the learner.

To capture the first aspect, we define a function $\hat{R} : \mathcal{S} \times \mathcal{Z} \to \mathbb{R}$ that represents an approximation of the instantaneous reward for each resource allocation, considering only the information available in the current state. If the QoS functions $\{f_m(\cdot)\}$ are known, we can consider the following function:

$$\hat{R}(\mathbf{s}, \mathbf{z}) = \sum_{m=1}^{M} \left[\sum_{i=1}^{z_m} f_m\left(\Delta_{q_m(i)}\right) - \sum_{j=z_m+1}^{L} f_m\left(\Delta_{q_m(i)} + 1\right) \right]. \tag{6}$$

Note that maximizing (6) may lead to suboptimal resource allocations, since $\hat{R}(\mathbf{s}, \mathbf{z})$ does not account for the long-term reward.

To model the second aspect, we consider that each DRL slot generates an experience sample, which requires ℓ packets (and as many resources) to be transferred to the learner. Due to memory limitations, we assume that the number of samples that can be buffered cannot exceed E. We hence define a second function $S(\mathbf{z}, e)$ that captures the effectiveness of the allocation $\mathbf{z}$ in transferring the $e \in \{0, \dots, E\}$ samples in the experience queue. In this paper, we define S in terms of the number of experience samples that we manage to transmit in the learning slot, defined by (17) in our use case, but, once again, this is a reasonable but arbitrary choice and other options might be more suitable for other scenarios.

The greedy strategy is then the solution to the following optimization problem:

$$\mathbf{z}^*(\mathbf{s}, \mathbf{z}, e) = \arg\max_{\mathbf{z}} \left(M^{-1} \hat{R}(\mathbf{s}, \mathbf{z}) + E^{-1} S(\mathbf{z}, e) \right). \quad (7)$$

If f_m is concave for all slices with a soft timing requirement, we can exploit the FIFO nature of the queue to provide a simple iterative solution, starting from the empty assignment and gradually assigning resources to either one of the slices or the learning process, depending on the value of the utility function. Fig. 2 shows a schematic of the full learning plane resource allocation strategy in a simple case with $M = 2$: at each time step, the node randomly selects either the DRL agent or (with probability $\rho(t)$) the greedy allocation, which reserves some resources for the learning plane.

3.3 Continual Learning

We assume the environment can be characterized by a set of parameters $\boldsymbol{\omega}$, which determine its stochastic dynamic. From time to time, this parameters can instantaneously change, making the environment non-stationary (in the DRL jargon, changing the task) and requiring the strategy to be updated in order to pursue the new task.

To cope with such a non-stationary context we proposed a CL strategy similar to the work in [15], but including considerations on the cost of learning. When a context-change is detected, say from $\boldsymbol{\omega}$ to $\boldsymbol{\omega}'$, we consider its significance by means of a distance function $\eta(\boldsymbol{\omega}, \boldsymbol{\omega}')$: if it is larger than a threshold η_{thr}, we consider the environment to be novel enough to warrant a retraining. On the other hand, if the change is smaller than the threshold, we implictly assume that the policy is close enough to the optimum that maintaining it is cheaper than running a new training phase.

We define the environment index $k \in \mathbb{N}$, which starts from 0 and is incremented at every significant change in the environment. Our solution maintains a record of the past environments and the respective learned policies, so that as the environment shifts into the new context $\boldsymbol{\omega}_{k+1} = \boldsymbol{\omega}'$, we can find the closest past environment:

$$j^* = \arg\min_{j \in \{0,\dots,k\}} \eta(\boldsymbol{\omega}_j, \boldsymbol{\omega}'). \quad (8)$$

If the previously experienced environment is close enough to the new one, i.e., $\eta(\boldsymbol{\omega}_{j^*}, \boldsymbol{\omega}') < \eta_{\text{thr}}$, we can apply the stored policy directly, relying on a short training phase with increased exploration rate and training probability $\rho(t)$ to adapt to the small change. If no environment in the memory is close enough, training needs to begin from scratch, with slower and more expensive training phase.

4 Network Slicing Use Case

To substantiate the approach on a practical but easy to analyze use case, we consider the resource allocation problem in a simple network slicing scenario. We assume a common communication link is used to transmit both the data packets generated by the users, which belong to two different network slices, and the pieces of information used to feed the learner. Time is divided in slots of constant duration τ, and in each slot the transmission channel can carry N orthogonal and identical resource blocks. The scenario fits the general model presented in the previous section, as the communication resources are shared between the data and learning planes. The full parameters for the scenario, which we will describe in this section, are given in Tab. 1.

TABLE 1: Use case and learning parameters.

Parameter	Symbol	Value
Communication system		
Number of subchannels	N	15
Slot time duration	τ	1 ms
Packet queue length	Q	1500
Packet size	L	512 B
Link capacity	C	7.68 MB/s
Learning plane		
Discount factor	γ	0.95
Learning queue length	E	1500
Packets required for each sample	ℓ	3
Initial learning slot probability	ρ_0	0.2
Final learning slot probability	ρ_f	0.01
Learning slot probability decay	σ	8×10^{-4}
Learning slot decay pace	H	1000
Queue pressure parameter	χ_1	1400

4.1 Communication System Model

We consider two slices, corresponding to the two types of data sources:

- Slice 1 is for bulky file transfer, for which we do not set any strict latency constraints. However, we want the system to have the highest possible reliability to ease the burden on the higher layers. As such, $f_1(T) = 1$ for all finite values of T, but the QoS is 0 if T is infinite (i.e., if the packet is dropped);
- Slice 2 is intended for interactive traffic, such as video conferencing or Virtual Reality (VR) traffic, with a strict latency deadline: packets need to be transmitted with a maximum latency $T^{(2)}_{\text{soft}}$. For the sake of simplicity, we assume that, after $T^{(2)}_{\text{soft}}$, the utility decreases linearly with time, dropping to 0 if the latency is higher than $T^{(2)}_{\max} \geq T^{(2)}_{\text{soft}}$, i.e., $f_2(x) = 1$ if $0 \leq x \leq T^{(2)}_{\text{soft}}$, and $f_2(x) = \max\left(0, 1 - (x - T^{(2)}_{\text{soft}})/(T^{(2)}_{\max} - T^{(2)}_{\text{soft}})\right)$ if $x > T^{(2)}_{\text{soft}}$.

We remark that, although these QoS functions are reasonable, they may not be the most appropriate to represent the considered slices. Since the purpose of this study is to gain insights on the cost of learning in dynamic systems, more than proposing a quantitative performance analysis of the use-case, we prefer these neatly-shaped functions that allow for a qualitative performance analysis while easing the interpretability of the results.

The number of active users in each slice is variable, making traffic non-deterministic. We consider a maximum number of active users $U_m \in \mathbb{N}$ for each slice $m \in \{1, 2\}$. Each user follows a on-off model, which can be modeled as a Gilbert-Elliott binary Markov chain with transition probability matrix $\mathbf{O}^{(m)}$. In state 0, the user does not transmit,

TABLE 2: Traffic Model Parameters

Env. Index	Policies		Slice 1 Params					Slice 2 Params					
	ω_{start}	ω_{end}	U_1	R_1	$\mathbf{O}^{(1)}$	$\mathbb{E}[U_1]$	U_2	R_2	$\mathbf{O}^{(2)}$	$\mathbb{E}[U_2]$	$T^{(2)}_{\text{soft}}$	$T^{(2)}_{\max}$
$\boldsymbol{\omega}_0$	$\theta \sim \mathcal{N}(0, 0.1)$	θ_{ω_0}	28	512kB/s	$\begin{pmatrix} 0.617 & 0.382 \\ 0.544 & 0.455 \end{pmatrix}$	11.561	5	512kB/s	$\begin{pmatrix} 0.156 & 0.843 \\ 0.763 & 0.236 \end{pmatrix}$	2.625	50ms	70ms
$\boldsymbol{\omega}_{12}$	θ_{ω_1}	$\theta_{\omega_{12}}$	15	512kB/s	$\begin{pmatrix} 0.158 & 0.841 \\ 0.218 & 0.781 \end{pmatrix}$	11.908	30	512kB/s	$\begin{pmatrix} 0.925 & 0.074 \\ 0.672 & 0.327 \end{pmatrix}$	2.976	50ms	70ms
$\boldsymbol{\omega}_{102}$	$\theta_{\omega_{23}}$	$\theta_{\omega_{102}}$	20	512kB/s	$\begin{pmatrix} 0.797 & 0.202 \\ 0.316 & 0.683 \end{pmatrix}$	7.810	83	512kB/s	$\begin{pmatrix} 0.949 & 0.050 \\ 0.547 & 0.452 \end{pmatrix}$	6.944	50ms	70ms
$\boldsymbol{\omega}_{110}$	$\theta_{\omega_{75}}$	$\theta_{\omega_{110}}$	9	512kB/s	$\begin{pmatrix} 0.696 & 0.303 \\ 0.156 & 0.843 \end{pmatrix}$	5.937	37	512kB/s	$\begin{pmatrix} 0.804 & 0.195 \\ 0.620 & 0.379 \end{pmatrix}$	8.870	50ms	70ms

while in state 1, it transmits packets of size L with a constant bitrate R_m.

The aggregate traffic generated by slice m is then represented by the number $u_m(t)$ of active users at time t, multiplied by R_m. We can then define a Markov chain over $u_m \in \{0, \dots, U_m\}$, with the following transition probabilities:

$$P(u_m(t+1) = v | u_m(t) = u) = \sum_{w=\max(0,u+v-U_m)}^{\min(u,v)} (O_{11}^{(m)})^w (O_{10}^{(m)})^{u-w} \times \binom{u}{w} \binom{U_m - u}{v - w} (O_{01}^{(m)})^{v-w} (O_{00}^{(m)})^{U_m-u-v+w}. \tag{9}$$

The expected traffic G_m from slice m can be computed as:

$$\mathbb{E}[G_m] = \frac{O_{01}^{(m)} U_m R_m}{O_{01}^{(m)} + O_{10}^{(m)}}. \tag{10}$$

On the other hand, the total channel capacity is simply:

$$C = \frac{NL}{\tau}. \tag{11}$$

With the values in Tab. 1, we obtain $C = 7.68$ MB/s.

Note that, based on the definition, slice 1 can only experience rejected packets, while slice 2 can have both rejected and dropped packets (if their age exceeds the deadline $T^{(2)}_{\max}$).

4.2 Learning Plane

In this part we define the two components of the learning plane, i.e., the DRL agent, which will assign resources during the DRL slots, and the greedy split approach, which manages resource allocation in the learning slots.

DRL agent settings: We use a Deep Q-Network (DQN) [40] for the agent, as the problem is simple enough not to require more advanced architectures.

We consider a simplified state: for each slice $m \in \{1, 2\}$, the input to the network is given by the following values:

- The number $q_m \in \{0, \dots, Q\}$ of packets in the queue;
- The minimum latency $T_m^{\min}$ for packets transmitted in the previous slot;
- The maximum latency $T_m^{\max}$ for packets transmitted in the previous slot;
- The average latency T_m^{avg} for packets transmitted in the previous slot;
- The number d_m of discarded (dropped or rejected) packets in the previous slot;
- The current number a_m of resource blocks allocated to the slice.

The values for each queue are contained in the tuple $\mathbf{s}^{(m)} = (q_m, T_m^{\min}, T_m^{\max}, T_m^{\text{avg}}, d_m, a_m)$, to which we add another parameter $\xi^{(m)}$, i.e., the difference in the utility for slice m if packets are not transmitted in the next slot. For slice 1, i.e., the latency-insensitive one, this corresponds to the expected number of rejected packets; for the second slice, it is only applicable if some packets are close to or over the soft deadline $T^{(2)}_{\text{soft}}$. If the head-of-line packets are close to $T^{(2)}_{\max}$, this can even lead to packet drops. We can define it as follows:

$$\xi^{(2)} = \sum_{i=1}^{q_m} f_2\left(\Delta_{q_2(i)}\right) - f_2\left(\Delta_{q_2(i)} + 1\right). \tag{12}$$

As the first slice does not have latency requirements, there are no equivalent parameters for it. All the input values are normalized to fit in the range between 0 and 1.

The input to the DQN is then given by $\mathbf{s}^{(m)}, \xi^{(m)}$ which corresponds to a total of 13 values; the training parameters are defined in Tab. 1.

For what concerns the action space, we denote by $\mathbf{a}_t = [a_1, a_2]$ the resource allocation vector during slot t. At each step, the DRL makes an action δ_t to change the resource allocation as:

$$\mathbf{a}_{t+1} = \mathbf{a}_t + \delta_t. \tag{13}$$

For the sake of simplicity and interpretability of the results, we admit only actions $\delta_t \in \{(1, -1), (0, 0), (-1, 1)\}$ that change the allocation to each slice of at most 1 resource block per step. The outputs of the DQN correspond to the estimated long-term value of selecting each δ_t, so the network only has 3 output values.

The full network architecture is given in Table 3[1].

Greedy split algorithm settings: We set the size of the experience sample queue $E = 1500$, and implement an early rejection policy. When a sample is generated, its rejection probability is equal to $\frac{e}{E}$, i.e., to the current pressure on the queue. Consequently, samples that find a full queue are always rejected, but sometimes samples that could fit in the

1. The complete implementation of the DQN agent and dynamic resource allocation is available at https://github.com/slahmer97/costoflearning

TABLE 3: DQN architecture.

Layer size		Activation function
Input	Output	
13	64	ReLU
64	32	ReLU
32	3	Linear

queue are dropped in favor of new experiences, avoiding too many correlated samples filling the queue.

The probability of selecting a slot as a learning slot decays linearly, starting from an initial value ρ_0 and gradually decaying to a value ρ_f: every H steps the learning rate is decreased by a constant value σ:

$$\rho(t) = \max\left(\rho_f, \rho_0 - \left\lfloor \frac{t}{H} \right\rfloor \sigma\right). \tag{14}$$

Finally, we consider the greedy allocation in the learning slots. As the first slice has no latency requirements, we consider allocating resources to it greedily only when the number of packets in the queue is higher than a threshold χ_1: in this way, we avoid packet rejections, but also leave more resources for learning plane and latency-sensitive packets.

We can then define the following estimated rewards:

$$\hat{R}_1(\mathbf{s}, \mathbf{z}) = \min(0, z_1 - \min(q_1 - \chi_1, N)); \tag{15}$$

$$\hat{R}_2(\mathbf{s}, \mathbf{z}) = \min(0, z_2 - \min(\xi_2, N)); \tag{16}$$

$$S(\mathbf{z}, e) = -\left(\min(e, N) - \left(N - \sum_{m=1}^{2} z_m\right)\right). \tag{17}$$

The minimum operation ensures that resources will not be allocated to a slice once the queue pressure is below the limit ξ_1 or all packets with a close deadline are served, respectively. We can define the following problem:

$$\mathbf{z}^*(\mathbf{s}, \mathbf{z}, e) = \arg\max_{\mathbf{z} \in \mathcal{Z}} S(\mathbf{z}, e) + \sum_{m=1}^{2} \hat{R}_m(\mathbf{s}, \mathbf{z}). \tag{18}$$

As the problem can easily be converted to an integer linear problem, we can easily solve it through iterative methods.

4.3 Continual Learning

In our environment, the statistics of the traffic change periodically every 500 seconds: the parameter vector $\boldsymbol{\omega}$ includes $\mathbf{O}^{(1)}$, $\mathbf{O}^{(2)}$, U_1, or U_2, and as a result, corresponding changes are made to the transition matrix $\mathbf{P}$. The policy for a given set of environmental parameters $\boldsymbol{\omega}$ is defined by the set of corresponding trained weights vectors in the DRL neural network, $\theta_{\boldsymbol{\omega}}$.

In the slicing task, the average traffic for each slice is enough to characterize a new environment, and we can identify each context with the vector $\boldsymbol{\omega} = (\mathbb{E}[U_1], \mathbb{E}[U_2])$. Additionally, we define the threshold η as the point at which we trigger this event. In other words, we only initiate a change event if the distance between two environments is greater than η. While the average may not accurately represent the environment due to potential variance changes with a constant average, in our specific scenario, the average proved sufficient in maintaining a system performance near the ideal one (as described in the following section). The implementation of a robust method for detecting changes in the environment is critical, as the failure to identify a true change or the detection of a false change can lead to a degradation in system performance. However, in this study, we focus on the efficiency-vs-effectiveness tradeoff of the learning and defer the study of advanced and reliable context-recognition problem to future research.

Following the strategy we outlined in Sec. 3-3.3, the weights of the neural network are chosen from the closest environment observed in the past. We can also make an additional consideration: if the offered traffic is decreasing for both slices, the previous policy will still obtain good results, as the new environment is substantially easier than the previous one. We then define a strict minority relation between vectors, so that $\mathbf{x} \prec \mathbf{y}$ if the two vectors $\mathbf{x}$ and $\mathbf{y}$ are the same length and each element of $\mathbf{x}$ is smaller than the corresponding element of $\mathbf{y}$:

$$\mathbf{x} \prec \mathbf{y} \Leftrightarrow |\mathbf{x}| = |\mathbf{y}| \wedge x_i < y_i,\ \forall i \in \{1, \ldots, |\mathbf{x}|\}. \tag{19}$$

We also employ the Euclidean distance to define $\eta(\boldsymbol{\omega}, \boldsymbol{\omega}')$ between two environments, so the centralized agent is updated as follows:

$$\theta_{\boldsymbol{\omega}_{k+1}} = \begin{cases} \theta_{\boldsymbol{\omega}_k}, & \text{if } \boldsymbol{\omega}_{k+1} \prec \boldsymbol{\omega}_k; \\ \theta_{\boldsymbol{\omega}_{j^*}}, & \text{if } \|\boldsymbol{\omega}_{k+1} - \boldsymbol{\omega}_{j^*}\|_2 < \eta_{\text{thr}}; \\ \theta \sim \mathcal{N}(0, 0.1), & \text{otherwise,} \end{cases} \tag{20}$$

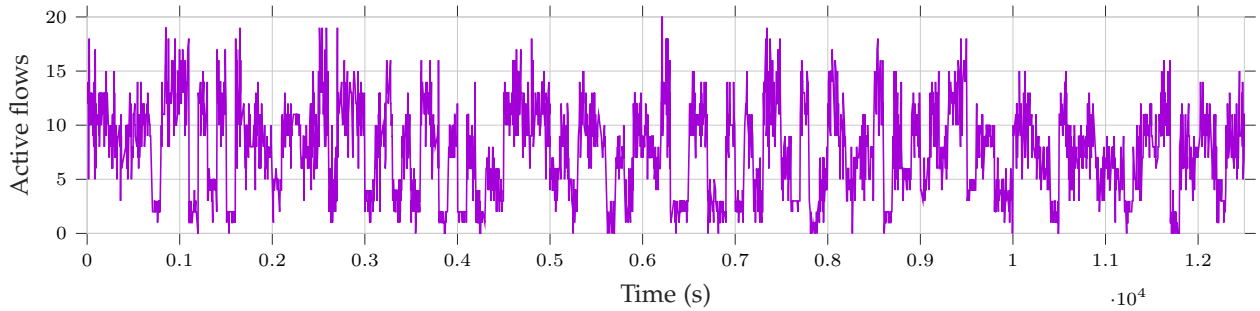
where $||\mathbf{x}||_2$ is the ℓ_2 norm of vector $\mathbf{x}$. After the new weight vector is selected, the algorithm temporarily increases both the training slot probability $\rho(t)$ and the exploration rate of the DRL agent, so that the new policy can be adapted to the new task.

5 Simulation Settings and Results

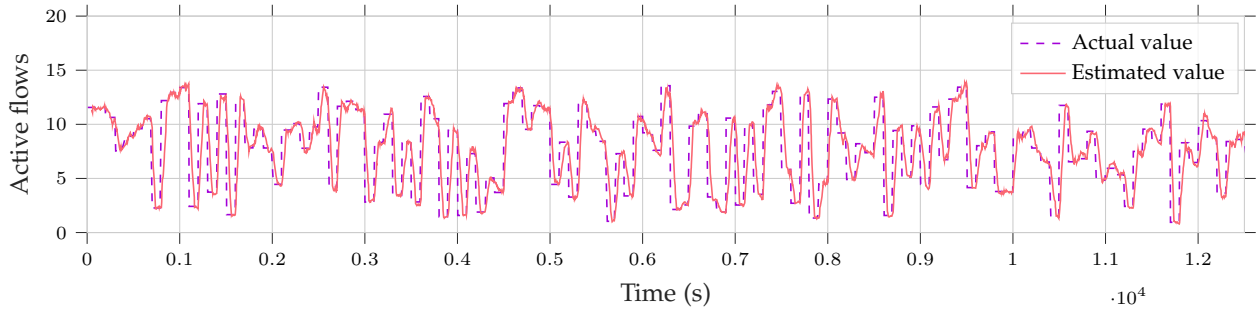
In this section, we present numerical findings that demonstrate the efficacy of the dynamic learning plane resource allocation policy in a non-stationary environment. To assess the proposed framework, we run the resource allocation for 64000 seconds, corresponding to 128 coherence periods lasting 500 seconds each. As each allocation step corresponds to 1 ms, this means that the environment is statistically stable for 5×10^5 steps, then abruptly transitions to a different behavior. The changes in the environment are produced using Algorithm 1. In the algorithm, we denote the probability of a user belonging to slice m being active as on_m, and the uniform distribution between a and b as $\mathcal{U}(a, b)$. The full parameters of the simulation model are given in Tables 1 and 2.

We consider four different benchmarks for the proposed scheme:

- *Out-of-band*: this scheme represents the ideal case in which training data is transmitted over a side channel with infinite capacity. This aligns with the common assumption in the literature of free training, and represents an upper bound for performance;
- *Frequency Division Multiple Access (FDMA)*: here we assume 1 resource block in each slot is reserved to the learning plane, while the other 14 resource blocks can be freely allocated to the users' slices;



(a) Instantaneous number of active flows.



(b) Smoothed number of active flows.

Fig. 3: Number of active flows over time for each coherence period in the first slice.

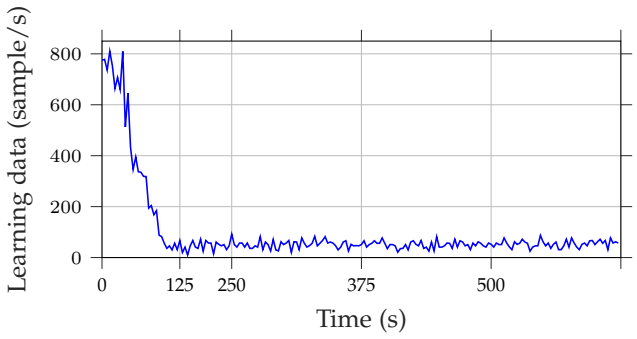


Fig. 4: Average number of forwarded learning samples per second.

- *Time Division Multiple Access (TDMA)*: we consider a time division between the learning and data planes, in which all available resources are allocated to the learning plane once every T_ℓ slots. We consider two cases, with $T_\ell = 10$ and $T_\ell = 100$.

In the following, we will also consider a normalized reward, equal to 1 if all packets are delivered with utility 1 (i.e., before $T_{\text{soft}}^{(2)}$ if they belong to slice 2) and 0 if all packets are dropped or rejected.

Fig. 3a shows the total number of active flows (i.e., of users transmitting data in the slot) in the first slice during the entire simulation time, while Fig. 3b reports a smoothed time average, along with the tracked number of active flows. We can clearly see that the smoothed average is tracked relatively well, so that any significant drift is detected promptly and dealt with by the CL scheme.

We can also consider the effect of the shared resources on both the learning and data planes: Fig. 4 shows how many experience samples are forwarded to the Cloud during the training process. Following the linear decay of $\rho(t)$, the number of new experience samples transmitted for training is initially very high, but decreases over about 70 seconds to reach the minimum, which is between 40 and 50 samples per second. This rate is high enough to guarantee that changes in the environment statistics are captured, but does not impact the final performance, as we will show in the following. Furthermore, we can analyze the impact of learning slots on the instantaneous reward by looking at the empirical Cumulative Distribution Function (CDF) of the reward penalty from using the greedy allocation, shown in Fig. 5: the reward loss is 0 in 40% of cases, and below 0.1 in 80% of cases. This means that the greedy allocation can still guarantee good performance in most cases, and as such, is a robust strategy for the learning slots.

We have explored the impact of three different initialization strategies on system performance: the proposed strat-

Algorithm 1 Environment Parameter Update

Output: $\mathbf{O}^{(1)}$, $\mathbf{O}^{(2)}$, $\mathbf{U_1}$, $\mathbf{U_2}$

while $\frac{\mathbb{E}[G_m]}{C} \notin [0.75, 1.1]$ **do**
  for i $\in [1, 2]$ **do**
    $\mathbf{O}^{(i)}[1][1] = \mathcal{U}(0.05, 0.95)$
    $\mathbf{O}^{(i)}[1][0] = 1.0 - \mathbf{O}^{(i)}[1][1]$
    $\mathbf{O}^{(i)}[0][0] = \mathcal{U}(0.05, 0.95)$
    $\mathbf{O}^{(i)}[0][1] = 1.0 - \mathbf{O}^{(i)}[0][0]$
  end for
  $\mathbf{on_0} = \frac{\mathbf{O}^{(0)}[0][1]}{(\mathbf{O}^{(0)}[0][1]+\mathbf{O}^{(0)}[1][0])}$
  $\mathbf{on_1} = \frac{\mathbf{O}^{(1)}[0][1]}{(\mathbf{O}^{(1)}[0][1]+\mathbf{O}^{(1)}[1][0])}$
  $\mathbf{U_1} = \texttt{random}\left(2, \left\lfloor \frac{14}{\mathbf{on_0}} \right\rfloor, 1\right)$
  $\mathbf{U_2} = \left\lfloor \frac{\max(\lfloor 15 - \mathbf{U_1 on_0} \rfloor, 1)}{\mathbf{on_1}} \right\rfloor$
  Compute the resulting $\mathbb{E}[G_m]$
end while
return $\mathbf{O}^{(1)}$, $\mathbf{O}^{(2)}$, $\mathbf{U_1}$, $\mathbf{U_2}$

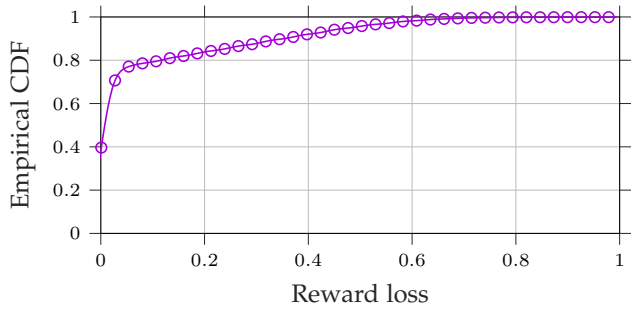


Fig. 5: Empirical CDF of the reward loss during learning slots.

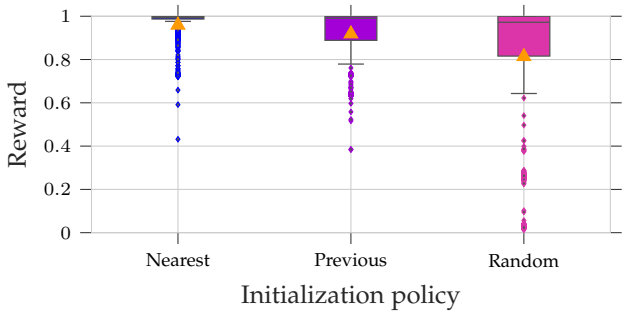


Fig. 6: Distribution of the performance with different initialization strategies after drift is detected.

egy, which uses the nearest recorded environment, provides a significant boost over keeping the previous environment or randomly selecting a memorized one, as Fig. 6 shows: the nearest environment selection improves CL by starting the DQN with weights that are already close to the correct ones, reducing the mistakes in the initial phases of the retraining (as shown by the limited number of outliers in the boxplot).

We also sampled four different coherence periods for the system (i.e., periods of time during which the context does not change), whose parameters are reported in Tab. 2. The reported index corresponds to the time of their appearance in the simulation. In all the selected environments, the load is greater than 0.945, i.e., the offered traffic is very close to the channel capacity, and in env. 12 and 110, the load is around 0.99. Fig. 7 shows the average reward for these environments, along with the packet drop rate for slice 1 (i.e., packets exceeding $T^{(2)}_{\max}$, which are dropped from the queue as their utility is 0) and the packet rejection rate for slice 2 (i.e., packets which find a full buffer and are discarded directly). The indices of the periods represent an incremental number of different environment seen by the CL agent: the first, with index 0, is the first to be seen, while there are 12 other periods between 0 and 12, and so on. Each period has the same duration, i.e., 500 seconds.

As a first observation, we note that the ideal out-of-band policy, which neglects the cost of learning, clearly outperforms those that reserve some resources to the learning plane, which confirms that the cost of learning is not negligible and needs to be accounted for when designing the resource allocation strategies, as we have done in our "Dynamic" scheme. The bar plots in Fig. 7a-d, in fact, show that our scheme can outperform the static FDMA and TDMA resource allocation strategies, almost reaching the same performance as the ideal out-of-band system. The only cases with an appreciable performance gap between our scheme and the out-of-band system are env. 12 and env. 110: as we remarked above, these are the most challenging ones, with a total load close to or over 99% of the nominal link capacity. In these limit cases, any learning policy that requires resources for the training will unavoidably determine the violation of the QoS requirements for some users, which further highlights the importance of the cost of learning in the system design.

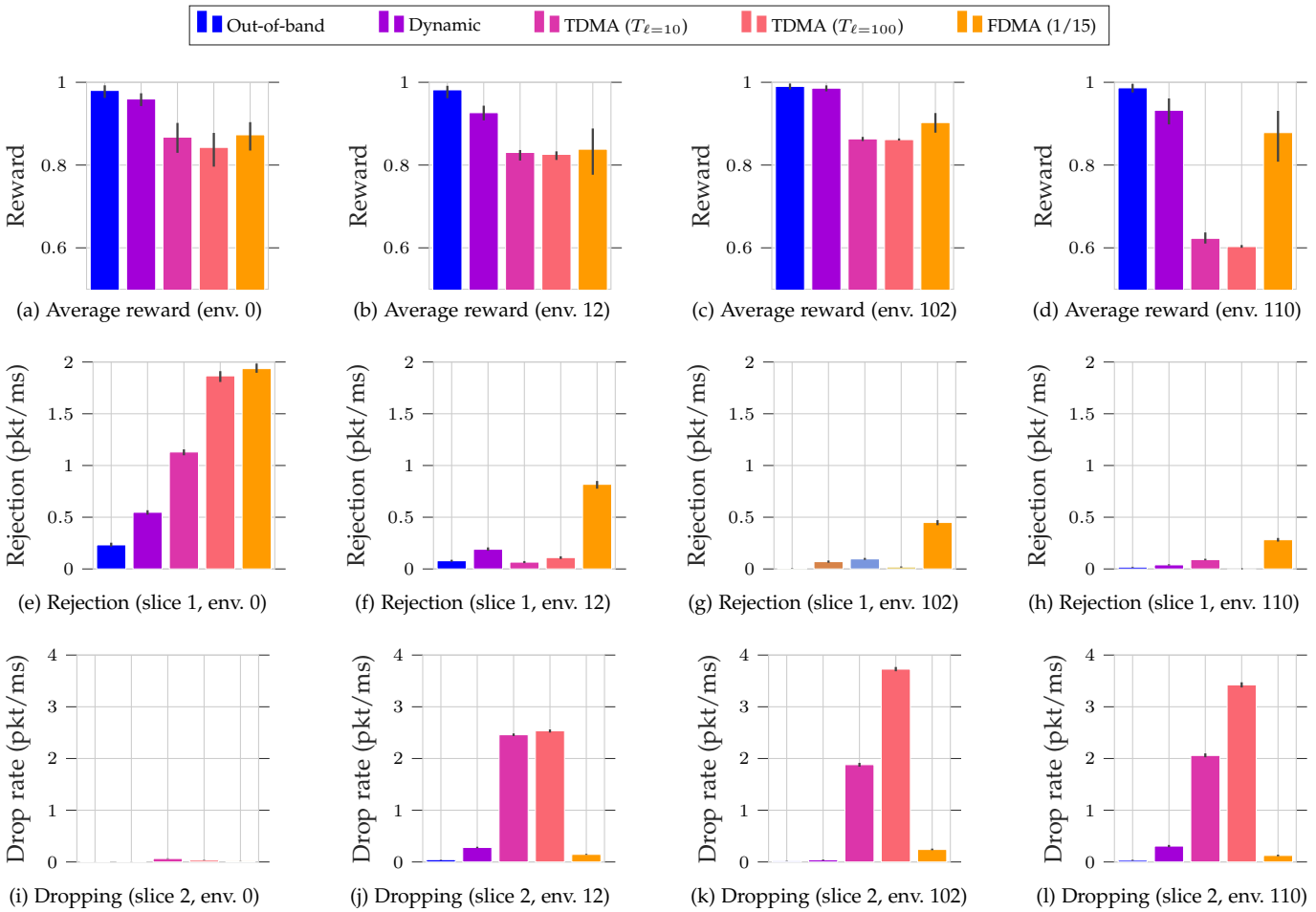
The relative simplicity of the system model we considered makes it possible to analyze in depth the choices made by the different schemes. From the bar plots in Fig. 7e to Fig. 7l we can observe that the FDMA and TDMA schemes tend to drop or reject a significant number of packets in all environments, while the ideal and dynamic ones manage to limit the number of unserved packets for both slices. Interestingly, even the ideal scheme drops a significant number of packets from slice 1 in env. 0, but performance is still high. We can explain this by considering Fig. 8, which shows the empirical CDF of the latency for packets in slice 2. Fig. 8a clearly shows that almost all packets have utility 1, i.e., are delivered before $T^{(2)}_{\text{soft}}$: in this case, all schemes tend to privilege slice 2, filling the queue in slice 1 more often. We should also consider that the learning agents start from scratch in environment 0, i.e., they have no pre-trained weights to start from, and we should expect a relatively large number of mistakes.

We can also see that the ideal and dynamic schemes have matching latency profiles in env. 102, as Fig. 8c shows, while the FDMA and TDMA schemes tend to transmit more packets with a latency close to $T^{(2)}_{\max}$. In environments 12 and 110, shown in Fig. 8b and Fig. 8d, respectively, the dynamic scheme drops more packets than the ideal one, and has a higher overall latency, but still outperforms the static allocation schemes. Interestingly, the two TDMA schemes tend to have better latency performance than the dynamic scheme in env. 12, but cannot improve the utility: the fraction of packets with a latency higher than $T^{(2)}_{\text{soft}}$ is the same for all three schemes, and the two TDMA ones drop a large number of packets, causing a significant performance difference. In this case, serving most packets from slice 2 as soon as they arrive is not advantageous, as it leads to worse performance overall for TDMA.

6 Conclusions and Future Directions

In this work, we have designed a dynamic resource allocation policy, which can mediate between the learning and data planes, controlling the trade-off between effectiveness and efficiency of DRL models. Unlike most works in the learning-based networking literature, we specifically consider the cost of learning, i.e., the resources required by the training process of a DRL agent, and show that our dynamic policy can outperform static schemes and, after a short transition phase, match the performance of an ideal system with an out-of-band learning plane. Furthermore, the adaptability of the scheme is demonstrated by applying it in a CL setting with environment changes, to which the dynamic scheme adapts extremely quickly.

Possible extensions of the work certainly include the adaptation of the scheme to more complex scenarios, with a larger number of resources and slices and more stringent QoS requirements, as those for Ultra-Reliable Low-Latency

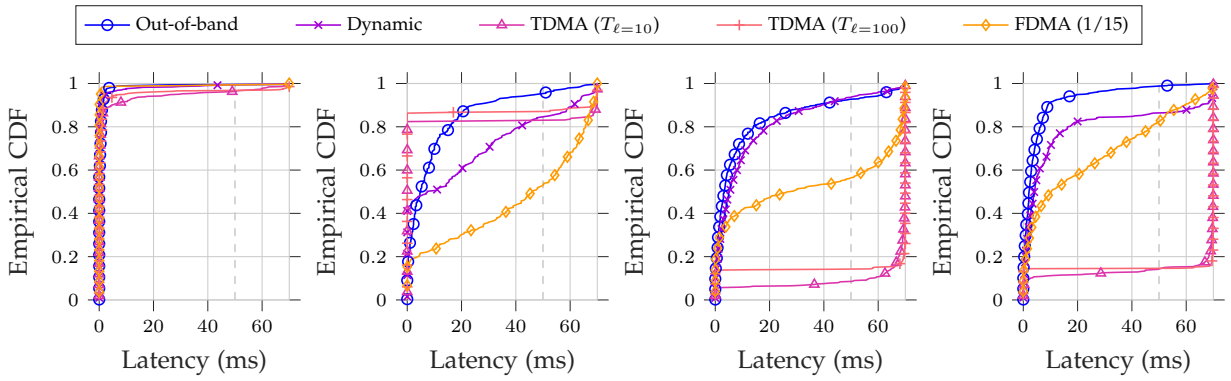


(a) Average reward (env. 0) (b) Average reward (env. 12) (c) Average reward (env. 102) (d) Average reward (env. 110)

(e) Rejection (slice 1, env. 0) (f) Rejection (slice 1, env. 12) (g) Rejection (slice 1, env. 102) (h) Rejection (slice 1, env. 110)

(i) Dropping (slice 2, env. 0) (j) Dropping (slice 2, env. 12) (k) Dropping (slice 2, env. 102) (l) Dropping (slice 2, env. 110)

Fig. 7: Performance of the schemes in four different sampled environments, measured by the average normalized reward, the packet rejection rate for slice 1, and the packet drop rate for slice 2.



(a) Env. 0 (b) Env. 12 (c) Env. 102 (d) Env. 110

Fig. 8: Empirical CDF of the latency for slice 2 in the four selected environments.

Communications (URLLC). However, even more interesting would be addressing some theoretical questions, such as the interplay between the cost of learning and active learning, which requires to select the most valuable samples to be transmitted in order to accelerate the training, particularly when resources in the learning plane are scarce. As mentioned, furthermore, the detection of context changes that trigger a retraining of the network is another open challenge. Finally, of particular interest is the design of meta-learning schemes that can learn when the resource allocation scheme needs to be retrained, balancing the potential performance improvement that could be brought about by a retrained policy and the cost to learn it, relative to the expected system coherence time.

REFERENCES

[1] K. B. Letaief, W. Chen, Y. Shi, J. Zhang, and Y.-J. A. Zhang, "The roadmap to 6G: AI empowered wireless networks," *IEEE Communications Magazine*, vol. 57, no. 8, pp. 84–90, 2019.

[2] M. Yang, Y. Li, D. Jin, L. Zeng, X. Wu, and A. V. Vasilakos, "Software-defined and virtualized future mobile and wireless networks: A survey," *Mobile Networks and Applications*, vol. 20, no. 1, pp. 4–18, 2015.

[3] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429–2453, 2018.

[4] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[5] H. Sami, H. Otrok, J. Bentahar, and A. Mourad, "AI-based resource provisioning of IoE services in 6G: A deep reinforcement learning approach," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3527–3540, 2021.

[6] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[7] M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying forgetting in classification tasks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 7, pp. 3366–3385, 2021.

[8] N. Shalavi, G. Perin, A. Zanella, and M. Rossi, "Energy efficient deployment and orchestration of computing resources at the network edge: a survey on algorithms, trends and open challenges," 2022. [Online]. Available: https://arxiv.org/abs/2209.14141

[9] I. Chih-Lin, "AI as an essential element of a Green 6G," *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 1, pp. 1–3, 2021.

[10] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on Mobile Edge Computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[11] F. Mason, F. Chiariotti, and A. Zanella, "No free lunch: Balancing learning and exploitation at the network edge," in *International Conference on Communications (ICC)*. IEEE, 2022, pp. 631–636.

[12] F. Mason, G. Nencioni, and A. Zanella, "Using distributed reinforcement learning for resource orchestration in a network slicing scenario," *IEEE/ACM Transactions on Networking*, 2022.

[13] S. Lahmer, F. Chiariotti, and A. Zanella, "The cost of learning: Efficiency vs. efficacy of learning-based RRM for 6G," in *International Conference on Communications (ICC)*. IEEE, 2023.

[14] X. Li, Y. Zhou, T. Wu, R. Socher, and C. Xiong, "Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting," in *International Conference on Machine Learning*. PMLR, 2019, pp. 3925–3934.

[15] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, 2016.

[16] D. Rolnick, A. Ahuja, J. Schwarz, T. Lillicrap, and G. Wayne, "Experience replay for continual learning," in *Proc. 33rd International Conference on Advances in Neural Information Processing systems (NeurIPS)*. ACM, 2019, pp. 350–360.

[17] D. Isele and A. Cosgun, "Selective experience replay for lifelong learning," in *Proc. AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[18] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 12, pp. 2935–2947, 2017.

[19] H. Ahn, S. Cha, D. Lee, and T. Moon, "Uncertainty-based continual learning with adaptive regularization," in *Proc. 33rd International Conference on Advances in Neural Information Processing systems (NeurIPS)*. ACM, 2019, pp. 4392–4402.

[20] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *Proc. 34th International Conference on Machine Learning (ICML)*, vol. 70. PMLR, 2017, p. 3987–3995.

[21] A. Mallya and S. Lazebnik, "Packnet: Adding multiple tasks to a single network by iterative pruning," in *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018, pp. 7765–7773.

[22] J. Serra, D. Suris, M. Miron, and A. Karatzoglou, "Overcoming catastrophic forgetting with hard attention to the task," in *Proc. 35th International Conference on Machine Learning (ICML)*. PMLR, 2018, pp. 4548–4557.

[23] J. Schwarz, W. Czarnecki, J. Luketina, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell, "Progress & compress: A scalable framework for continual learning," in *Proc. 35th International Conference on Machine Learning (ICML)*. PMLR, 2018, pp. 4528–4537.

[24] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 5149–5169, 2021.

[25] C. Finn, A. Rajeswaran, S. Kakade, and S. Levine, "Online meta-learning," in *Proc. 36th International Conference on Machine Learning (ICML)*. PMLR, 2019, pp. 1920–1930.

[26] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. Freitas, and J. Sohl-Dickstein, "Learned optimizers that scale and generalize," in *Proc. 34th International Conference on Machine Learning (ICML)*. PMLR, 2017, pp. 3751–3760.

[27] R. Houthooft, Y. Chen, P. Isola, B. Stadie, F. Wolski, O. Jonathan Ho, and P. Abbeel, "Evolved policy gradients," in *Proc. 32nd International Conference on Advances in Neural Information Processing Systems (NeurIPS)*. ACM, 2018, pp. 5405–5414.

[28] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4780–4789.

[29] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. 36th International conference on machine learning (ICML)*. PMLR, 2017, pp. 1126–1135.

[30] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 24–35, 2019.

[31] J. Xu and Z. Zhu, "Reinforced continual learning," in *Proc. 32nd International Conference on Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31. ACM, 2018, pp. 907–916.

[32] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Proc. 31st International Conference on Advances in neural information processing systems (NeurIPS)*. ACM, 2017, pp. 4080–4090.

[33] A. Antoniou, H. Edwards, and A. Storkey, "How to train your MAML," in *Proc. 7th International Conference on Learning Representations (ICLR)*, 2019.

[34] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," in *Proc. 30th International Conference on Advances in Neural Information Processing Systems (NeurIPS)*. ACM, 2016, pp. 3988–3996.

[35] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," in *Proc. International Conference on Learning Representations (ICLR)*, 2019.

[36] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-reinforcement learning of structured exploration strategies," *Advances in neural information processing systems*, vol. 31, 2018.

[37] L. Yang and A. Shami, "A lightweight concept drift detection and adaptation framework for IoT data streams," *IEEE Internet of Things Magazine*, vol. 4, no. 2, pp. 96–101, 2021.

[38] A. Bifet, "Adaptive learning and mining for data streams and frequent patterns," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 55–56, 2009.

[39] E. R. Faria, I. J. Gonçalves, A. C. de Carvalho, and J. Gama, "Novelty detection in data streams," *Artificial Intelligence Review*, vol. 45, pp. 235–269, 2016.

[40] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.