Adaptive Control Policies for Core Network Autoscaling with Meta-RL

Shantanu Verma*, Abhishek Gupta*, Jatinder Singh*‡,, Jose Manuel Sanchez Vilchez†, Guillaume Fraysse†
*Orange Innovation Networks, Gurgaon, India †Orange Research, Paris, France
{shantanu.verma, abhishek3.gupta, jose2.sanchez, guillaume.fraysse}@orange.com ‡jsrp0607@gmail.com

***Abstract*—The problem of Core Network (CN) autoscaling is evaluated with Deep Reinforcement Learning (DRL) agents. Complex CN Network Functions (NFs) flows and data requirements for Reinforcement Learning (RL) agent training make the entire training process slow. Trained models can become inefficient when workloads change over time. Meta-RL has been introduced and one of its goals is to train models that can adapt to unseen tasks. This paper introduces CoreNetTwin, a Network Digital Twin (NDT) of a CN that can train RL models and compare their performance in a time-efficient manner. Using a NDT allows to train a model faster and enables training on synthetic data. A methodology to build a dataset to train a RL model with a NDT to help the model learn the dynamics of the environment is introduced. The CN autoscaling problem is then mapped to a Meta-RL setting characterized by a Meta-RL task. The performance of the RL^2 Meta-RL algorithm is evaluated, using Proximal Policy Optimization (PPO) as the *base-learner*, and compared to a baseline. The results show that CoreNetTwin enables the rapid evaluation of RL algorithms and that RL^2 can better adapt to unseen patterns of traffic, maintaining a level of performance equivalent to a model trained for a specific task.**

***Index Terms*—Reinforcement Learning, Autoscaling, Packet Core Network Functions, Meta-RL, Network Digital Twin**

I. Introduction

Precisely scaling the resources used on their infrastructure according to the workload allows telecommunications operator to reduce their costs, power consumption and carbon footprint. The scaling of NFs can be performed at different layers, on the baremetal infrastructure itself or, since the introduction of Network Function Virtualization (NFV), on the virtual resources like Virtual Machines (VMs) or containers.

The Network Management Community has adopted Machine Learning (ML) controllers to multiple management tasks. RL is one of the ML paradigms where optimal control policy could be learned via interactions with the environment and help to achieve dynamic control over multiple tasks.

Previous work [1] using RL to address the autoscaling of NFs shows that training a RL model on live traffic is a lengthy process that can last for several days [1]. The work presented here introduces a first version of a NDT, called CoreNetTwin, that can be used to train a RL model in several minutes, making it easier to develop and tune models.

Another challenge faced when training a RL model to scale NFs is that the workload can change over time. This paper shows a 2-step solution to address this problem. First it explains how to build a workload for the training process that helps the model learn possible patterns in the workload. Then it explores the use of Meta-RL algorithms, a class of algorithms that learn to learn and can adapt to unseen patterns of traffic.

The CN autoscaling problem is to scale up or down the number of available instances of NFs according to the workload, e.g., Session Management Function (SMF). In the scope of this paper's experiments only control-plane traffic is considered.

Section II defines RL and Meta-RL. Then Section III provides a state of the art of CN autoscaling, RL for Network Management as well as Meta-RL. The contributions of this paper are detailed in the other sections:

1) CoreNetTwin, a NDT of an open-source CN, is introduced in Section IV along with the process of creating a training workload to improve the trained model,
2) Formulation of the CN autoscaling problem as Meta-RL tasks is done in section V,
3) Section VI details the experimental results for the evaluation of the performance of the RL^2 Meta-RL algorithm for the CN autoscaling problem against a classic DRL algorithm as the baseline.

II. RL and Meta-RL definitions

This section introduces RL and Meta-RL. It gives some details on the algorithms used in the experiments detailed in Section V: Dueling Double Deep Q-networks (D3QN), that is used as the baseline, as well as RL^2 and PPO that are used together in the Meta-RL experiments.

A. Reinforcement Learning

RL consists of an agent exploring an environment for Horizon (T) time steps with some reward r_t at each step t. This interaction is rooted in a series of actions a_t applied by the agent according to a policy π_θ parameterized by θ that leads the environment to new states S_t. The goal of the RL agent is to obtain the maximum discounted ($\gamma \in [0, 1]$) return η through the sequence of interactions over the horizon (T). This horizon can either be finite or infinite without a predefined endpoint. $\eta(\pi_\theta) = \mathbb{E}[\sum_{t_0}^{T} \gamma^t r_t(s_t, a_t)]$ or $\eta(\pi_\theta) = \mathbb{E}[\sum_{t_0}^{\infty} \gamma^t r_t(s_t, a_t)]$

A RL problem can be modeled as a discrete time discounted Markov Decision Process (MDP) that can be defined as:

$$M = (S, A, P, r, \rho_0, \gamma, T) \tag{1}$$

where P represents the state transition probability and ρ_0 depicts the initial state distribution [2].

978-3-903176-75-1 ©2025 IFIP

The overall training goal of any RL agent is to derive an optimal policy $\pi^*(a|s)$ describing the action$(a) \in A$ (*Action space*) taken in each state $s \in S$ (*State space*). RL can be categorized into two different approaches based on how the optimal policy is computed [3]:

1) **Value-based methods** that estimate a value function. π^* is computed from the value functions by evaluating either the state values V:

$$V(s) = \mathbb{E}[\sum_{t_0}^{\infty} \gamma^t r_t | s_0 = s]$$

or the state-action values Q:

$$Q(s,a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a]$$

In the particular case of Dynamic Programming or Monte Carlo method, the state value functions are estimated and an optimal policy is derived from the state values as:

$$\pi^*(s) = \arg\max_a \sum_{a'} P(s'|s,a)[r(s,a,s') + \gamma V(s')]$$

and from action values as:

$$\pi^*(s) = \arg\max_a \sum_{a'} Q(s,a)$$

in State Action Reward State Action (SARSA) [4] or Q-learning method. These methods are useful in simple and discrete (state/action) environments but harder to compute in complex and continuous environments.

2) **Policy Gradient (PG) methods** that help to directly optimize a parameterized policy instead inferring from state or action values. Policy is modeled as probability distribution over action space, $P(a|s;\theta)$. PG methods find the parameterized policy $\pi(\theta)$ by computing the gradient ascent of the objective function that leads to the direction in which parameter θ is to be changed to maximize the cumulative total return. The policy gradient $\hat{g}$ is:

$$\hat{g} = \hat{\mathbb{E}}_t[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t]$$

where $\pi(\theta)$ is a stochastic policy and $\hat{A}_t$ is an advantage function estimator at timestep t that scales the gradient and log probability to steer the direction toward the best actions. The advantage is a relative measure of goodness of action compared to other actions in the same state:

$$\hat{A}_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k} - V(s_t) \tag{2}$$

The PG can be obtained by an optimization loss function or an objective function L^{PG} :

$$L^{PG} = \hat{\mathbb{E}}_t[log\pi_\theta(a_t|s_t)\hat{A}_t] \tag{3}$$

Some of the most common PG methods are REINFORCE [5], Actor-Critic [3], Trust region policy optimization (TRPO) [6] and Soft Actor-Critic (SAC) [7]

We use DRL as Deep Neural Networks (NNs) are good function approximators that can enable generalization across similar states and tasks which is crucial for Meta-RL. The next two sections introduces the two DRL algorithms D3QN and PPO.

B. D3QN

Q-learning belongs to the class of value-based and off-policy RL methods. D3QN [8] is an architectural enhancement over Double DQN (DDQN) [9] and Deep Q-Network (DQN). It splits the stream of fully connected layers in a typical DQN into value $V(S;\theta,\beta)$ stream and advantage $A(s,a;\theta,\alpha)$ stream. Contrary to DQN and DDQN where the network directly outputs the Q-value of each action, in D3QN the Q-function is estimated by the linear combination of value function $V(s)$ and advantage function $A(s,a)$ as below:

$$\begin{aligned} Q(s,a;\theta,\alpha,\beta) &= V(s;\theta,\beta) + (A(s,a;\theta,\alpha) \\ &\quad -\frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha)) \end{aligned}$$

Equivalent form of cf. Eq. (2) is used, i.e., $A(s,a) = Q(s,a) - V(s)$. The target Q-value is computed using the online network to select the best action, like in DDQN, and the target network computes the Q-value of the selected action.

Experience replay buffer is a key component in off-policy methods, used during training to approximate the Q-function. It stores the agent-environment interactions (s,a,s',r) in a buffer, collectively referred to as *experiences*. Mini-batches are sampled from this buffer to train the network. Mainly, two kinds of replay buffer are discussed in the literature, distinguished by their sampling technique - *Uniform* and *Prioritized* [10] experience replay buffer. Experience replay improves sample efficiency by breaking temporal dependence from the data and making data closer to being independent and identically distributed (i.i.d.).

C. PPO

PPO [11] is an actor-critic based PG method. It is sample efficient, stable, and scalable compared to other actor-critic methods. It is designed to address the instability issue with the vanilla policy gradients where large updates can deviate the policy significantly from optimal region making recovery difficult. PPO makes use of surrogate clip objective instead of the aforementioned loss function L^{PG} (cf. Eq. (3)) to ensure stable updates:

$$L^C = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$

where: $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ The probability ratio $r_t(\theta)$ quantifies the change in the new policy compared to the older one and the clipping function allows for control over policy update by restricting it within the range $[1-\epsilon, 1+\epsilon]$. It uses advantage estimation, namely Generalized Advantage Estimate (GAE) that balances the bias-variance tradeoff using parameter $\lambda \in [0,1]$:

$$\hat{A}_t = \sum_{k=0}^{T-t-1} (\gamma\lambda)^k \delta_{t+k}$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$. The advantage $\hat{A}_t$ scales the policy updates. If $A > 0$ the likelihood of the action increases, and it decreases if $A < 0$. It ensures that the policy shift towards better than expected actions and away from worse ones. Effectively PPO optimizes the following loss function :

$$L_t^{PPO} = \hat{\mathbb{E}}_t[L_t^C - c_1 L_t^{VF} + c_2 \mathcal{H}(\pi_\theta)] \tag{4}$$

where c_1, c_2 are the coefficients, L^{VF} is the critic loss function, $(V_\theta(s_t) - V_t^{targ})^2$ where V_t^{targ} is computed from empirical return or advantage estimate, and $\mathcal{H}(\pi_\theta)$ is an entropy bonus to ensure proper exploration. Reduction and stabilization of the value of entropy is indicative of policy convergence and actions that could be trusted.

PPO starts by sampling data from the environment for a specific Horizon (T) as per current policy and uses it to optimize the surrogate objective, cf. Eq. (4), using the gradient ascent method. PPO, being on-policy, collects fresh data everytime according to the latest policy and discards it after optimization step.

D. Meta-RL

Meta-Learning [12] has been introduced with the goal of quick adaptation to new tasks for which they were not trained initially. The ability to adapt quickly to new tasks is facilitated by leveraging the prior knowledge on similar tasks. Few shot classification [13] is a classical example of Meta-Learning where a model is trained on a limited set of examples to learn a shared structure which enables rapid adaptation to unseen tasks. While classical learning techniques are trained on a single task using large number of training examples and expected to perform well on the same, yet to generalize well on structurally similar but unseen tasks they often requires training from scratch. In contrast, humans are able to adapt to new tasks by using prior knowledge gained from unrelated previous tasks, Meta-Learning is built upon the same idea. The motivation of using Meta-Learning often arises from the lack of sufficient training data, when the cost of retraining is very high or the learning method is sensitive to the change in training data.

A regular supervised learning method uses *predefined meta-knowledge* ω [14] to guide the learning of optimal parameters that minimize/maximize it's objective function. ω can be the choice of optimization algorithm (Stochastic Gradient Descent (SGD), Adam ...), weight initialization or regularization, etc. In contrast, Meta-Learning doesn't assume that meta-knowledge is given rather it learns the same via *meta-learner*. Thus it is commonly said, *learning how to learn*.

Meta-RL is a particular case of Meta-Learning applied to a RL problem. It is viewed as a bi-level optimization problem, where the outer-loop or *meta-learner* learns the algorithm that is being optimized in the inner-loop called as *base-learner*. While the *base-learner* is concerned with the task specific learning, *meta-learner* accumulates knowledge from range of tasks presented to the *base-learner*.

Generally to train a Meta-RL algorithm, an MDP is sampled from a distribution $\mathcal{P}(\mathcal{M})$ and the RL algorithm (*base-learner*) is optimized on it. This interaction of *base-learner* with the tasks is known as trial, a trial can consist of multiple episodes which together forms a *meta-trajectory*. In case of continuous control, a single trial can be said as a meta-trajectory. While parameters of *meta-learner* (meta-parameters) are updated between the trials based on the performance of *base-learner*. Hence, we wish to find optimal meta-parameters ω^* representing meta-knowledge:

$$\omega^* = \arg\max_{\omega} \ \mathbb{E}_{\mathcal{M}\sim p(\mathcal{M})} \left[\mathbb{E}_{\tau\sim\pi_{\mathcal{M},\theta}}[R(\tau)]\right]$$

Such that it enables a RL agent (*base-learner*) to quickly adapt and maximize reward across different MDPs rather than a single one. During test, the meta-knowledge gained by *meta-learner* is what enables the RL agent to adapt to new tasks requiring only a few iterations.

It is essential to note that *Generalization* [15] refers to the capability of a ML model to perform well on unseen tasks that were not used for the training but usually train and test tasks are assumed to come from the same distribution. Additionally, *Adaptation* is quantified by the amount of data required to update a ML model to be able to generalize on new tasks. Meta-Learning or Meta-RL algorithms are designed to incorporate the capability of *rapid adaptation* in regular learning methods resulting in *generalization across tasks* and improved sample efficiency during test time.

E. Types of Meta-RL techniques

According to [16], few-shot, multi-task Meta-RL techniques are of special interest that can be classified into 3 categories based on inner-loop adaptation mechanism or meta-knowledge representation:

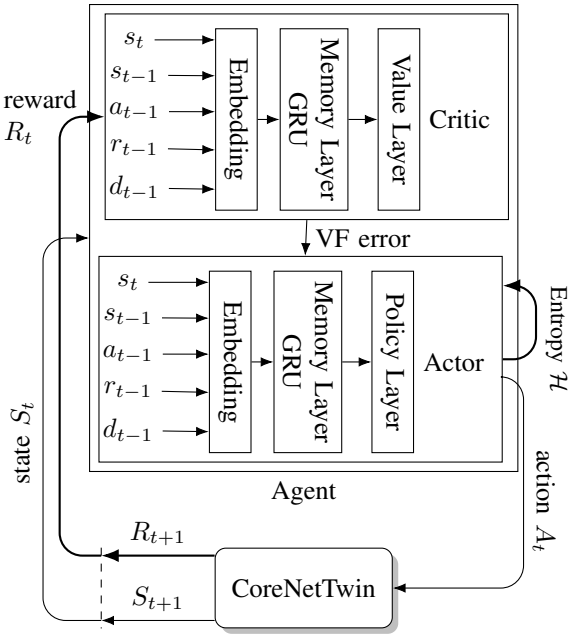


Fig. 1: RL^2 algorithm with PPO as the *base-learner*

- Black box methods: in these methods the policy is modeled via recurrent learners and meta-knowledge is captured in the internal memory (e.g., hidden state in RNN). Adaptation occurs through changes in internal memory rather than updates in the weights. Examples include RL^2 [2] and SNAIL [17].
- Gradient based: learns the shared initialization parameters. Meta-knowledge is represented as a set of initial parameters that can be fine tuned via gradient descent. Adaptation is enabled by the parameter updates. e.g., MAML [18] and REPTILE [19]
- Task inference or latent variable based: learns the latent task embedding from agent interactions using encoders. The embedding captures meta-knowledge about task distribution and the policy is conditioned on it. During test, the agent can identify the nature of the task within few interaction with new task that enables quick adaptation. Examples include PEARL [20] and VariBAD [21].

In this work we focus on the black-box method RL^2.

F. RL^2

Duan et al. [2] introduced RL^2. This Meta-RL algorithm has two components called *fast* and *slow*. The *fast* component stores the experiences in the activations of a Recurrent Neural Network (RNN), enabling the fast task specific adaptation, whereas the *slow* component is the RL algorithm that shapes and updates the RNN weights across a distribution of tasks. It acquires a general adaptation strategy allowing RNN to effectively handle new tasks. Fig.1 shows the schematic implementation of RL^2 with PPO as *base-learner* and Gated Recurrent Unit (GRU) as *meta-learner* that is used for the RL^2 experiments described in Section V. It also outlines the interaction of RL^2 with the environment, i.e. CoreNetTwin described in Section IV. The implementation of the agent resembles that of PPO where actor-critic architecture is followed with the difference being that the policy layer and the value layer are preceded by the meta-learner. In RL^2, a policy is represented as a RNN, both the actor and critic receive the tuple $(s_t, s_{t-1}, a_{t-1}, r_{t-1}, d_{t-1})$ as input (where d_t is a flag set to 1 when the episode is done), this is preprocessed by the embedding layer where the discrete actions are one-hot encoded, then all the inputs are combined into a vector. The outputs of actor and critic (RL^2) agent are conditioned on the context vector (meta-knowledge accumulated by the GRU), and the current observations. The actor models the categorical probability distribution, and the action is sampled from it. The entropy bonus $\mathcal{H}$ is derived from the same probability distribution that is added to the actor's loss, encouraging policy exploration. The chosen action is executed in the environment, resulting in a new state S_{t+1} and a reward R_{t+1} as feedback to the agent. The critic essentially outputs the value estimate based on S_t and makes use of rewards to form an advantage estimate colloquially VF error, which is later used by the actor to guide policy updates.

III. State of the Art

The possibilities of RL when it comes to the realm of network management seem to be unbounded. As a matter of fact, RL is indeed considered by 3GPP standard consortium as one of the ML techniques for the management of 5G and beyond networks [22]. Several solutions [23]–[25] have been suggested to use RL in one of the RAN Intelligent Controllers (RIC) of the O-RAN alliance[1]. RL has been applied to Software-Defined Networking (SDN), whose key feature is that the forwarding decision are centralized in controllers. For e.g., [26] proposed Meta-RL to deal with the dynamic resource allocation and load distribution in SDN. This Meta-RL approach adapts its learning policy to the current task by analyzing the load congestion. It is based on a Bayesian Network that updates the posterior probability distribution from the SDN characteristics like the congestion load.

Another example, proposed by [27] is a Flow-based Service Time optimization based on DRL to cope with Rule placement problem due to the Ternary Content Addressable Memory constraints in SDN switches. Galliera et al. in [28] proposed an actor-critic RL algorithm, namely MARLIN to control and optimize congestion in distributed environments. The key idea was to use RL to cope with heterogeneous workloads in distributed environments. Later on, they embedded MARLIN in a framework [29] to be able to apply their RL solution on Tactical Environments such as Satellite communications and UHF radio links, proving RL versatility when it comes to dealing with heterogeneous conditions. [30] proposed an Inverse RL method to optimize the way power is allocated in wireless communication systems. Inverse RL automatically derives reward functions from expert policies, avoiding the bias of manual design. RL is also considered in the field of Cybersecurity. [31] proposed DRL as an intrusion detection mechanism for smart vehicular networks, whereas [32] proposed a scenario-agnostic zero-trust defense Meta-RL approach which was proven to be generic enough to a portfolio of defense strategies. Their approach is conceived for POMDP (Partially Observable MDP) and the Meta-RL technique is based on a gradient-based adaptation and more specifically a SGD algorithm.

RL is also a solution explored for resource allocation. Majumbar et al. in [33] proposed the usage of RL for two different purposes: (i) to autoscale Virtual CPU resources in network slices, and (ii) to distribute intelligence for autoscaling. The work done in [1], [34] tackles the autoscaling problem for core NFs using DRL on Magma. This work goes indeed beyond this to compare multiple more efficient RL algorithms and introduces a NDT that allows simpler and faster experiments before deploying them to an actual network.

Training a RL model on actual NFs is a complex and lengthy process, and takes longer due to several attempts required for the tuning of hyperparameters. To make this process simpler and faster, the de facto standard for the training of the RL

[1]https://www.o-ran.org/

model is the Open-Source Gym framework, now Gymnasium[2], originally introduced by OpenAI in 2016[3]. It provides multiple *environments* that simulate the behavior of cyber-physical systems.

RL has been adopted as an optimizer of the location of different networking functions in cloud environments. Santos et al. used RL for two main purposes in complex microservice-based applications in Kubernetes: (i) for an efficient auto-scaling [35], and (ii) for microservice deployment as shown in [36]. In [35], the authors introduced the gym-hpa environment for the autoscaling of microservice-based applications. It does not however specifically mimics the behavior of NFs and focuses on Kubernetes services, whereas our solution specifically targets NFs and is agnostic of the infrastructure which can also be VM-based. [37] proposed a Gym-based toolkit named Optical RL-Gym to apply RL in a flexible manner to problems related to optical networks, such as assigning resources in a Wavelength Division Multiplexing (WDM), known as Routing and Wavelength Assignment (RWA).

Building upon the concept of Digital Twin (DT) [38], the notion of NDT has been introduced [39] and Standard Development Organizations (SDOs) have started working on creating standards for them, e.g. IETF [40]. The goal of a NDT is to mimic the behavior of a network and enable several applications like troubleshooting, network planning, anomaly detection, etc. Having a single NDT that enables all applications is an ongoing challenge. Recent work has outlined a network sampling scheme with zoom-in/zoom-out operations to build accurate and scalable NDT for large-scale 5G/6G networks [41]. Instead of tackling this larger problem, the solution presented here provides a Gym-based environment that mimics the behavior of core NFs in the context of training a RL model for the autoscaling problem.

IV. CoreNetTwin: a Network Digital Twin for Core Network autoscaling

This section describes the experimental environment CoreNetTwin, a NDT based on the Gym framework. The hyperparameters are detailed and some feedback on how to tune them is given along with the values selected for the experiments. CoreNetTwin is then used in Section VI for the evaluation of the performance of the Meta-RL algorithm RL^2 introduced in Section II.

A. CoreNetTwin: a Network Digital Twin environment

CoreNetTwin is a Gym-based experimental testbed for the evaluation of RL algorithms which is a NDT of Magma [42], an open-source operator-grade packet CN. Magma packages the 3GPP NFs of LTE and 5G core in a single component called the Access Gateway (AGW).

We run our Physical Twin (PT) on a cloud-based environment. User Equipments (UEs) and Radio Access Network (RAN) are simulated using S1APTester[4] alongside with custom-made scripts to generate and load balance a workload across multiple instances. Each session attaches a UE for a random duration, modeled with a log-normal distribution. The average duration of sessions is 3 minutes, which is the average duration observed on the live network for Voice calls. Worloads are detailed in the next section.

In order to understand the behaviour of the AGW, a workload is run for few hours leveraging these scripts and the S1APTester. The impact of the workload on the AGW is then analyzed. We use the standard metrics provided by Magma but also one additional metric: the memory consumption of the Mobility Management Entity (MME) application running on the AGW that we collect with a custom-made workflow.

Correlation between metrics was used to select the metrics for the modelling of CoreNetTwin. It showed that the *attach rate* of UEs and the *memory usage* of the MME process have the highest correlation with dropped sessions, which is the metric that is the closest to the Quality of Service (QoS) in the considered use case.

Section V describes the mapping of the RL environment to the CN autoscaling problem and details how CoreNetTwin metrics are used as variables by the RL algorithms.

*1) **Motivation**:* As mentioned in [1], the total time to train a DRL algorithm for 10k steps on Magma vEPC can take up to an entire week. This is due to (i) long simulated traffic sessions ($\approx$ 3 minutes) and (ii) inherited delay in the NFs provisioning. Consequently, RL based CN autoscaling suffers from slow interaction cycles, creating bottlenecks in hyperparameter and NN architecture search as well as reward tuning. The aggregated time required can be difficult to manage in a project and may result in sub-optimal models

*2) **Modelling of the NDT**:* The NDT of Magma is designed as a Gym environment, following the principles of an RL environment. A Magma AGW instance is represented by a Python class where the properties or state of an AGW is represented by attributes. The AGW simulates workloads and simulates the value of the observed metrics. The metrics which represents the behavior of AGW are modeled mathematically and they are detailed in Section V-A. Observation of load tests on Magma shows that the attach rate is the main reason why sessions can't be established [1] and that a single AGW can handle approximately 3.33 new sessions per second. Accordingly CoreNetTwin considers that all the sessions above this threshold cannot be established because of the overhead. The probability of a crash is also simulated and increases with the attach rate.

B. Training workload description

The model's performance and capacity for generalization are significantly influenced by the training workload.

In the CN autoscaling problem the workload has a direct impact on the attach rate, which in turn impacts other metrics like the number of dropped sessions or whether the system crashes. Training an agent with a workload closely approximating the real-word situations exposes it to a wide range of

[2]https://gymnasium.farama.org/

[3]https://openai.com/index/openai-gym-beta/

[4]https://github.com/magma/S1APTester

state-action combinations prepares it to perform effectively on unseen data.

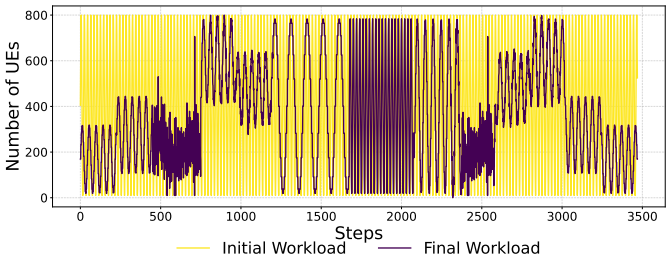


Fig. 2: sinewave and multi-pattern training workloads

The initial workload that was considered used a sinusoidal function (cf. Fig. 2), as per [1], to match the natural daily cycles observed in most digital services, where user activity follows predictable everyday patterns, with very limited number of connections at night and peaks during mornings and afternoons. To increase the variety, several steps were taken to create a **multi-pattern** workload:

- modify the shape of the sinusoid to have a flat curve for the high and low values of the number of UEs to simulate period where no scaling is needed,
- alternate amplitude and frequency of the sinusoid, to add more variety to the *attach rate* metric and to the min and max number of VMs required,
- append a subset of a production traffic dataset (taken from [43]).

This hybrid approach incorporates the required dynamic variations while maintaining the workload's overall structure. Fig. 2 shows final **multi-pattern** workload. Fig. 3 compares the performance of the D3QN algorithm when trained on these two different workloads. The evaluation is performed on the same workload than for the experiments described in V. The figure illustrates the evaluation results for the number of sessions dropped at different steps. Initially, both training workloads exhibit high session drops, exceeding 300. However, after approx 250k steps, the performance with the final (multi-pattern) workload stabilizes, maintaining near zero session drops whereas, the initial workload continues to show instability, with session drops persisting beyond 250k steps. Results show that tuning the training workload improved the performance of D3QN which was able to scale the platform to avoid most dropped sessions.

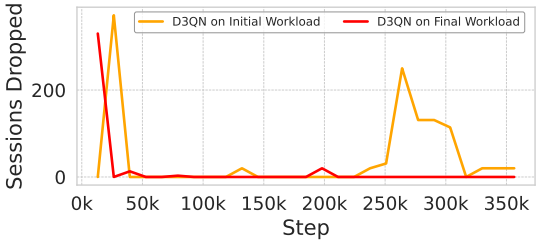


Fig. 3: Impact of the training workload on D3QN performance

C. Hyperparameter tuning

The performance of the aforementioned algorithms depends on a few hyperparameters (HPs). Tables I and II describe the HP values used in the experiments in the next section. HP tuning is carried out using Optuna [5].Furthermore, only for D3QN, these values were manually tuned to lower the *dropped sessions* D metric.

TABLE I: HPs For D3QN

HP	D3QN
Initial LR	1e-5
Exploration Steps	4000
Mini Replay Buffer Size	4000
Target Netw. Update Freq.	1000
Initial β	0.628
β Inc. per Step	0.001
PER α	0.6
Discount (γ)	0.99
Batch	32

TABLE II: HPs For RL^2

HP	PPO	RL2
Horizon (T)	284	12
Episodes per Trial (n)	*N/A*	22
Adam step $\times 10^{-5}$	2.95	21.9
Mini Batch	4	60
Num. epochs	10	5
Clipping parameter ϵ	0.1	0.1
VF coeff. c1	1	0.5
Entropy coeff.	0.01	0.01
Discount (γ)	0.99	0.99
GAE (λ)	0.95	0.95
Units	64	64 (GRU)
Gradient clip	-	5

Table I shows the values for the HPs for D3QN. Table II shows the values of the HPs for both algorithms used in the experiments described in Section VI. For PPO, the policy is updated using the *Mini-batch* size. The *clipping parameter* is used to avoid the large policy update step size. The Value Error function coefficient *VF coeff.* c_1 and *Entropy Coeff.* c_2 help to optimize the surrogate losses, cf. Eq. (4). *Discount* γ and *GAE* λ parameters help to estimate the advantage $\hat{A}_t$. *Units* is the number of hidden units of the NN layers. *Gradient clip* manages the exploding gradient problem during training of NN. Following early experiments, only one HP required tuning for RL^2, the Step per Meta-episode (SPM), i.e., after how many steps the *meta-learner* is updated from the *base-learner*. Tuning the other HPs show no improvement of the performance and the selected values are common ones. PPO and RL^2 both use a fixed learning rate.

V. Problem Description and Experiment Setup

This section shows how the CN autoscaling problem can be modeled as a discrete control problem where NFs can be scaled-up or scaled-down depending on resource requirements.

A. Core Network platform as a RL environment

The components of a RL system can be mapped with the autoscaling problem variables:

State space. The state of the CN platform is continuous and designed as a set of 4 NFs metrics including (i) Memory Usage (M) of the MME process of the AGW, (ii) number of simultaneously connected UEs (U), (iii) the attach rate R and (iv) the number V of running instances or VMs. The ranges of possible values of these metrics during the experiments are listed in Table III.

[5]https://github.com/optuna/optuna

TABLE III: State Space Of CoreNetTwin

State Metrics	Range
Memory Usage (M)	0-512 (MB)
Connected UEs (U)	0-250 (per NF)
Attach rate (R)	0-5 (per NF)
Count of NFs (V)	0-100 (count)

Action space. It is discrete with 3 possible actions: *scale-up* ($+1$ NF instance), *scale-down* (-1 NF instance), and *do-nothing*.

Reward function is designed to guide the agent toward guaranteeing QoS by minimizing the *dropped sessions* while using as few resources as possible. It observes the **state** as well as other metrics like if any crash has occured.

The reward is computed sequentially according to the order mentioned in Eq. (5), with mutually exclusive conditions evaluated in a prioritized manner. The reward is in the range $[-1, 1]$, the higher the better.

The ideal VMs (V_i) is calculated based on instances needed to handle the current load efficiently, maintaining 70% utilization per VMs and adding a small offset of 1.25 to account for uncertainties and overhead. If the system crashes, a fixed penalty of (-1) is applied to discourage actions that lead to system failure. Dropped sessions are penalized logarithmically to scale the penalty based on severity. Overprovision occurs when the number of active VMs exceeds a computed ideal VMs count.

To ensure optimal performance and avoid resource saturation, there is a 70% utilization cap on the total capacity. If the service performance is within this acceptable limit (≤ 0.7), the reward is proportional to how close the service performance is to 0.7 and if it exceeds it, a penalty is applied based on the deviation from the ideal value.

$$r = \begin{cases} -1 & \text{if } C = True \text{ else} \\ -log(D) & \text{if } D > 0 \text{ else} \\ 1 - \frac{|V - V_i|}{V_m} & \text{if } V_i < V \text{ else} \\ 1 - |0.7 - M| & \text{if } M \leq 0.7 \text{ else} \\ -(1 - M) & \text{otherwise} \end{cases} \quad (5)$$

Where C is Crash, D is Dropped Sessions and V_m is Maximum number of VMs.

The CN autoscaling problem can be modeled as a discrete control problem and can be solved by time-step MDPs with continuous states and discrete actions, cf. Eq. (1).

B. CN autoscaling as a Meta-RL problem

Task is sampled as the CN with workloads at different clock times. CN represents a different MDP at clock time with variable workload and a *base-learner* has to learn a optimal policy to solve these workload requirements.

Meta-task is to generalize the learning of *base-learner* of a task across the different workload patterns or network situations so that the agent can quickly adjust the policy accordingly with a few training steps to service the requirement.

Episode is a finite length sequence of n steps. During an episode the *base-learner* interacts with the environment with actions as per current policy, environment states, and immediate rewards. Episode length in Meta-RL training is generally lower than for RL training.

Trial is composed of multiple meta-episodes where RL^2 hidden states access the multiple states within the trial that accelerate the learning process of the *base-learner*. The objective of RL^2 is to optimize the policy maximum expected return over the trial. The hidden states are reset after the completion of trials but the learning is stored in the weights of RNN network of RL^2 layers.

VI. Results and Discussion

This section details the results of the experiments run on CoreNetTwin, the Gym-based NDT described in Section IV.

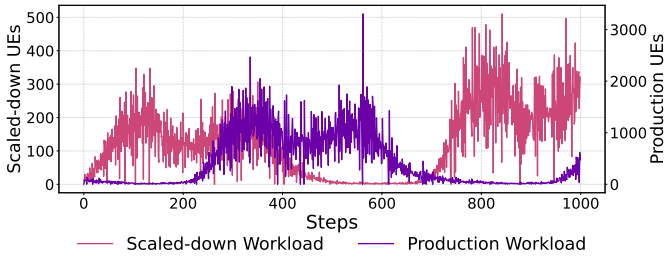


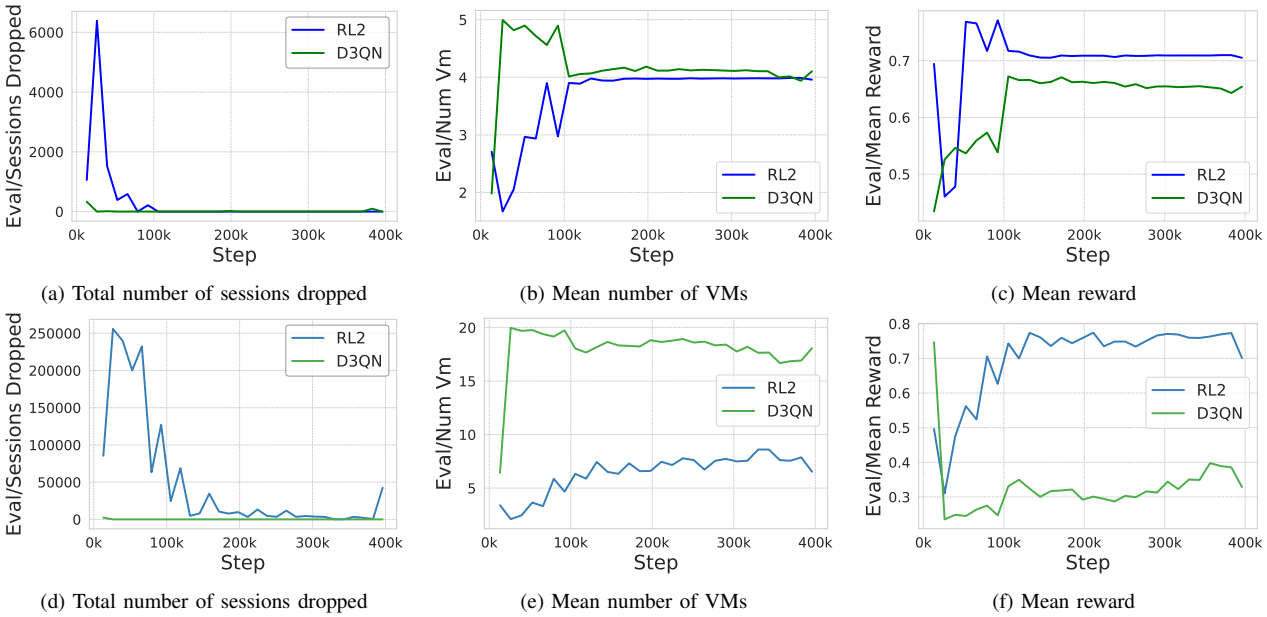
Fig. 4: The evaluation workloads: Production and Scaled-down

For the experiments, D3QN is used as a baseline, using a *Prioritized Experience Replay* buffer. It is appropriate for an environment with a discrete action space like the one described above. Then it is compared with the Meta-RL RL^2 algorithm with PPO as *base-learner* since RL^2 requires a PG-based algorithm as *base-learner*.

Training Process. The PPO and RL^2 agents are trained with the **multi-pattern** workload and configured with the HPs detailed in Section IV-B.

Evaluation Process The trained models are then evaluated against two different workloads that are shown in Fig. 4. A *production* workload, which is based from data collected on a live network from one location of Orange in France [43], and a *scaled-down* workload, which is calculated from the first to have a similar shape but a lower maximum number of connected UEs U, closer to the value of U in the training data. A subset of this same data was removed and integrated into the training dataset as described in IV-B. These 2 workloads allow to compare the performance on similar patterns but with different scales of traffic, trying to mimic unseen peak days. One evaluation is performed every 13200 steps. Each evaluation lasts for 7200 steps and metrics are averaged over this duration. Each point on the graph represents the mean value of the metric over all these 7200 steps. Fig. 5a-5f show the experimental results of both D3QN and RL^2 algorithms when trained models on the **multi-pattern** workload and evaluated on these two workloads.

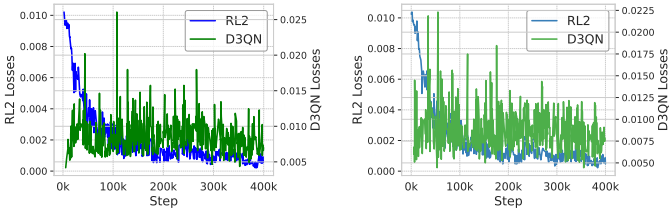
Fig. 5a-5c show the performance of both algorithms when evaluated on the *scaled-down* workload. In this test, we can see that both algorithms converge to a model that is able to preserve the QoS, i.e. no dropped sessions after 8 evaluations



(a) Total number of sessions dropped (b) Mean number of VMs (c) Mean reward

(d) Total number of sessions dropped (e) Mean number of VMs (f) Mean reward

Fig. 5: Results of D3QN and RL^2 evaluation on *scaled-down* (top), and *production* (bottom) workloads during experiments



(a) on *scaled-down* (b) on *production*

Fig. 6: Loss of D3QN and entropy loss of RL^2 during training

(or 105k steps) for RL^2 and after 2 evaluations for D3QN. Overall RL^2 uses 4% less VMs to handle the load : around 3.97 VMs when D3QN uses 4.12 in average. It also achieves a Reward $\approx 7.5\%$ higher which is expected since the Reward and the number of VMs used are correlated per definition, cf. Eq. (5). The performance of D3QN here is consistent with prior work [1] where it was evaluated on Magma, showing that CoreNetTwin is able to behave like Magma in the context of this usage.

Fig. 5d-5f show the performance of both algorithms against a different workload, called *production*, which has a shape similar to the previous one but with a higher number of connected UEs U. Here D3QN does not perform as well as before. It largely avoids *dropped sessions* but consumes a large number of resources (VMs) to do so and never learns to reduce the resource usage. Meanwhile RL^2 learns to use significantly less VMs, around 7 when D3QN uses 18 in average. It also manages to reduce the number of dropped sessions over time albeit in the tests is not able to reduce it to 0. This is also visible when comparing the mean reward of both algorithms: RL^2 consistently has a higher reward and reaches a high average reward around 0.75 when D3QN reaches average rewards between 0.24 and 0.35.

Fig. 6 shows the loss of both algorithm during training. It shows that the entropy loss of RL^2 is decreasing steadily while the loss D3QN shows that it is not able to improve the model further.

These results showcase that a Meta-RL algorithm like RL^2 is able to adapt better to some types of unseen traffic compare to a traditional DRL algorithm like D3QN. It is able to learn from different datasets, here workloads, to compute a model that can adapt to unseen data. The results also show that when used on data that is closer to the training data, it is also able to converge albeit slower than D3QN.

RL agents are known to be data-hungry, slow to train, complex to manage in production which limits their real world applicability. In telecom setting, NFs scaling depends on dynamic workload which can change with geography, time and service provider. Meta-RL provides a promising avenue by enabling generalization across tasks, paving the way for agents that are scalable, adaptive, interoperable and closer to production-grade given the training scenarios sufficiently represent real-world conditions.

VII. Conclusion and Future Works

This paper introduces the use of Meta-RL to address the autoscaling of NFs according to the workload. The experimental results show that the RL^2 model is able to adapt to an unseen workload when the DRL algorithm D3QN fails to do so. This is only a first step, other Meta-RL algorithms have been introduced since which could also be explored. RL, DRL and Meta-RL are interesting techniques that could be applied to other network use cases, future work includes the identification of these use cases.

The performance of the RL^2 algorithm is evaluated on CoreNetTwin, a NDT which is also introduced in this article.

Experiments conducted for this work showed that training a model on a NDT requires a significant work on the data that is fed to the model so that it can learn the largest possible number of action-space combinations. Currently it is a basic NDT that allows to mimic only the behavior of Magma, which is an all-in-one box CN. Work is ongoing to improve it to emulate all the NFs of a CN, using the Open-Source Open5GS CN as the PT, for the training of RL models.

REFERENCES

[1] J. Singh, S. Verma *et al.*, "Autoscaling Packet Core Network Functions with Deep Reinforcement Learning," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, 2023, pp. 1–6.

[2] Y. Duan, J. Schulman *et al.*, "Rl^2: Fast reinforcement learning via slow reinforcement learning," *arXiv preprint arXiv:1611.02779*, 2016.

[3] R. S. Sutton and A. G. Barto, "Policy gradient methods," in *Reinforcement Learning: An Introduction*. MIT Press, 2018, ch. 13.

[4] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering Cambridge, UK, 1994, vol. 37.

[5] R. Williams, "A class of gradient-estimation algorithms for reinforcement learning in neural networks," in *Proceedings of the International Conference on Neural Networks*, 1987, pp. II–601.

[6] J. Schulman, S. Levine *et al.*, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[7] T. Haarnoja, A. Zhou *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[8] Z. Wang, T. Schaul *et al.*, "Dueling Network Architectures for Deep Reinforcement Learning," in *Proceedings of The 33rd International Conference on Machine Learning*. PMLR, 2016, pp. 1995–2003.

[9] H. v. Hasselt, A. Guez *et al.*, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, p. 2094–2100.

[10] T. Schaul, J. Quan *et al.*, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[11] J. Schulman, F. Wolski *et al.*, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[12] J. Vanschoren, "Meta-learning: A survey," *arXiv preprint arXiv:1810.03548*, 2018.

[13] O. Vinyals, C. Blundell *et al.*, "Matching networks for one shot learning," *Advances in neural information processing systems*, vol. 29, 2016.

[14] T. Hospedales, A. Antoniou *et al.*, "Meta-learning in neural networks: A survey," 2020. [Online]. Available: https://arxiv.org/abs/2004.05439

[15] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4, no. 4.

[16] J. Beck, R. Vuorio *et al.*, "A survey of meta-reinforcement learning," 2023. [Online]. Available: https://arxiv.org/abs/2301.08028

[17] M. Nikhil, R. Mostafa *et al.*, "A simple neural attentive meta-learner," *ICLR*, 2018.

[18] C. Finn, P. Abbeel *et al.*, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 1126–1135.

[19] A. Nichol, J. Achiam *et al.*, "On first-order meta-learning algorithms," 2018. [Online]. Available: https://arxiv.org/abs/1803.02999

[20] K. Rakelly, A. Zhou *et al.*, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in *International conference on machine learning*. PMLR, 2019, pp. 5331–5340.

[21] L. Zintgraf, K. Shiarlis *et al.*, "Varibad: a very good method for bayes-adaptive deep rl via meta-learning," *Proceedings of ICLR 2020*, 2020.

[22] 3GPP, "Ts 28.908," 3GPP, Tech. Rep., 2024. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3965

[23] K. Boutiba, M. Bagaa *et al.*, "On enabling 5G Dynamic TDD by leveraging Deep Reinforcement Learning and O-RAN," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, 2023, pp. 1–3.

[24] H. Lee, Y. Jang *et al.*, "O-RAN AI/ML Workflow Implementation of Personalized Network Optimization via Reinforcement Learning," in *2021 IEEE Globecom Workshops (GC Wkshps)*, 2021, pp. 1–6.

[25] F. Rezazadeh, L. Zanzi *et al.*, "A multi-agent deep reinforcement learning approach for ran resource allocation in o-ran," in *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2023, pp. 1–2.

[26] A. Sharma, S. Tokekar *et al.*, "Meta-reinforcement learning based resource management in software defined networks using bayesian network," in *2023 IEEE 3rd International Conference on Technology, Engineering, Management for Societal impact using Marketing, Entrepreneurship and Talent (TEMSMET)*, 2023, pp. 1–6.

[27] M. Jiménez-Lázaro, J. Berrocal *et al.*, "Flow-based service time optimization in software-defined networks using deep reinforcement learning," *Computer Communications*, vol. 216, pp. 54–67, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366423004814

[28] R. Galliera, A. Morelli *et al.*, "Marlin: Soft actor-critic based reinforcement learning for congestion control in real networks," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, 2023, pp. 1–10.

[29] R. Galliera, M. Zaccarini *et al.*, "Learning to sail dynamic networks: The marlin reinforcement learning framework for congestion control in tactical environments," in *MILCOM 2023 - 2023 IEEE Military Communications Conference (MILCOM)*, 2023, pp. 424–429.

[30] R. Zhang, K. Xiong *et al.*, "Inverse Reinforcement Learning Meets Power Allocation in Multi-user Cellular Networks," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2022, pp. 1–2.

[31] Z. Wang, D. Jiang *et al.*, "A Deep Reinforcement Learning based Intrusion Detection Strategy for Smart Vehicular Networks," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2022, pp. 1–6.

[32] Y. Ge, T. Li *et al.*, "Scenario-Agnostic Zero-Trust Defense with Explainable Threshold Policy: A Meta-Learning Approach," in *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2023, pp. 1–6.

[33] S. Majumdar, S. Schwarzmann *et al.*, "Distributed intelligence for automated 6g network management using reinforcement learning," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, 2024, pp. 1–4.

[34] X. Long and G. Fraysse, "Safe rl for core network autoscaling," in *2024 20th International Conference on Network and Service Management (CNSM)*. IEEE, 2024, pp. 1–7.

[35] J. Santos, T. Wauters *et al.*, "gym-hpa: Efficient auto-scaling via reinforcement learning for complex microservice-based applications in kubernetes," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, 2023, pp. 1–9.

[36] J. Santos, M. Zaccarini *et al.*, "Efficient microservice deployment in kubernetes multi-clusters through reinforcement learning," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, 2024, pp. 1–9.

[37] C. Natalino and P. Monti, "The optical rl-gym: An open-source toolkit for applying reinforcement learning in optical networks," in *2020 22nd International Conference on Transparent Optical Networks (ICTON)*, 2020, pp. 1–5.

[38] F. Tao, H. Zhang *et al.*, "Digital twin in industry: State-of-the-art," *IEEE Transactions on industrial informatics*, vol. 15, no. 4, pp. 2405–2415, 2018.

[39] P. Almasan, M. Ferriol-Galmés *et al.*, "Network digital twin: Context, enabling technologies, and opportunities," *IEEE Communications Magazine*, vol. 60, no. 11, pp. 22–27, 2022.

[40] C. Zhou, H. Yang *et al.*, "Network digital twin: Concepts and reference architecture," *Internet Engineering Task Force, Internet-Draft draft-irtf-nmrg-network-digital-twin-arch-05*, 2024.

[41] M. Kellil, S. B. H. Said *et al.*, "Addressing the scalability of network digital twins: A network sampling approach," in *2024 20th International Conference on Network and Service Management (CNSM)*. IEEE, 2024, pp. 1–7.

[42] Magma. [Online]. Available: https://magmacore.org/

[43] A. Diamanti, J. M. S. Vilchez *et al.*, "Lstm-based radiography for anomaly detection in softwarized infrastructures," in *2020 32nd International Teletraffic Congress (ITC 32)*. IEEE, 2020, pp. 28–36.