Safe RL for Core Network autoscaling

Xinqi Long*, Guillaume Fraysse*
*Orange Innovation, France
xinqi.long@student-cs.fr, guillaume.fraysse@orange.com

***Abstract*—Core Network autoscaling is a critical scalability challenge, where traditional methods and existing advanced Reinforcement Learning (RL) approaches often overlook safety considerations leading to exploration of unsafe actions during training. This paper explores Safe Reinforcement Learning (Safe RL) as a solution to tackle these safety challenges. This paper formulates autoscaling as a Safe RL problem to integrate safety constraints. Then autoscaling methodologies are developed by optimizing Safe RL algorithms through delicate function design and hyperparameter tuning. Finally, Safe RL models are evaluated on a Gym-based simulation environment and the Magma Open-Source Core Network platform. Results confirm that Safe RL models have a safer behavior during exploration and can help real-world applicability of RL.**

I. Introduction

As networks grow in size and complexity, it becomes increasingly difficult to manage all of their different elements effectively and adapt them quickly to changing conditions and demands. The service requirements are dynamic, and the network state and traffic varies due to stochastic arriving requests with different Quality of Service (QoS) requirements (e.g., tolerated delay). Network Function Virtualization (NFV) allows to leverage the flexibility offered by virtualization and Software Defined Networking (SDN), and significantly reduces the time needed to scale out (resp. in) to increase (resp. decrease) the resources according to the workload of the Network Functions (NFs) [1]. Core Network autoscaling in telecommunications requires adaptive solutions to efficiently manage fluctuating traffic demands. Traditional methods often struggle to adapt swiftly, leading to inefficiencies and potential service disruptions. RL offers a promising approach by automating decision-making based on real-time feedback. However, applying RL in telecom poses challenges due to the critical need for safety and reliability. Safe RL integrates safety constraints into the learning process to ensure QoS requirements and prevent potential failures, thereby making it suitable for autoscaling. In this paper, we present several key contributions to advance Safe RL for Core Network autoscaling:

- Modeling Core Network autoscaling as a Safe RL problem in Section II. Our methodology includes visualization and Hyperparameter Tuning tools to aid in understanding and optimizing the performance of RL algorithms.
- Evaluation of Safe RL Algorithms on a Gym-based simulation environment in Section IV.
- Evaluation on the Magma Open-Source Core Network in Section VI.

A comprehensive State of the Art on Safe RL is detailed in Section III.

II. Problem statement

This section first defines the Core Network autoscaling problem. It then introduces RL and how it can be used to address this problem and finally Safe RL is defined.

A. Core Network Autoscaling

Core Network autoscaling is one of the most important scalability problem for the core network in NFV and can be summarized as: autoscaling the cloud resources(e.g. Virtual Machines (VMs)) allocated to NFs based on the dynamic workload. It's a waste if too many resources are allocated when the traffic is low, and allocating too few resources is hard to cope with sudden traffic spikes and may impact the QoS for subscribers or even lead to crash. In the autoscaling problem, the network is expected to be able to optimize the resources allocation to satisfy the demands with QoS requirements while minimizing the operation cost for the Operator.

The scaling mode can be proactive or reactive. Proactive scaling refers to the capability of predicting the future workload in order to schedule the necessary resources beforehand. The prediction of the traffic state may help the agent to get a better performance. Unlike proactive scaling, reactive scaling responds directly to real-time changes in traffic or system conditions as they occur, which aligns with the growing trend of applying RL methods to optimize decisions in real-time. A related topic is the autoscaling of the cloud infrastructure. Several papers about the application of RL for cloud autoscaling mentioned are covered in the survey. Few papers cover specifically the Core Network autoscaling problem. Nguyen et al. [2] investigate the application of Deep Reinforcement Learning (DRL) for scaling UPF instances that are packed in the containers of the Kubernetes container-orchestration framework in 5G/6G Core. In paper [1], a scaling method for packet core NFs using DRL is proposed, which learns how to handle workload and improving the QoS by reducing the number of dropped sessions. This work focuses on the horizontal scalability of virtualized Evolved Packet Core (EPC) NFs as shown in Figure 1, and it automatically scales the platform depending on the user traffic.

The Core Network autoscaling problem encapsulates the intricate task of dynamically adjusting the number of VMs or containers in response to fluctuating network traffic. It is facing the challenge of addressing the trade-offs between cost reduction, QoS preservation, and system stability. Overall, the

978-3-903176-66-9 ©2024 IFIP

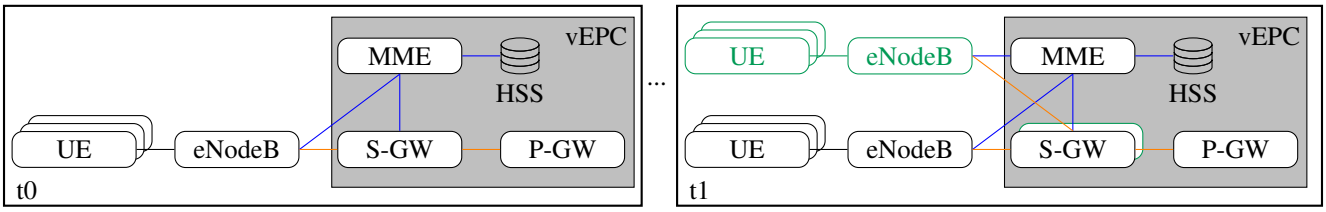


Fig. 1: Problem description for autoscaling of NFs for LTE networks

challenge of autoscaling is to find an efficient and effective way to scale the resources to NFs in order to meet the demands while also minimizing cost with guaranteed QoS and avoiding disruptions to service.

For the scaling problem in NFV, one promising approach is RL. In this section, we delve into the foundational concepts of RL and its extension, Safe RL.

B. Reinforcement Learning

RL is a branch of Machine Learning (ML) that enables an agent to learn and make decisions through trial and error by interacting with an environment [3]. The agent gets an observation of the state from the environment and then takes an action, and the environment can be changed by this action. The agent also gets feedback from the environment in the form of a reward, which measures how well the agent is doing in its current situation. The agent's ultimate aim is to maximize the total rewards it collects over time, which is often referred to as the return.

Recently, Deep Learning (DL) enables RL to address decision-making problems that were once too complex, especially in environments with high-dimensional state and action spaces [4]. DRL is a branch of ML that combines deep Neural Networks (NNs) with RL algorithms to learn from complex and dynamic environments [5]. DRL can handle high-dimensional and nonlinear problems and adapt to changing situations without requiring explicit rules or models. DRL has been successfully applied to various domains such as games, robotics and self-driving cars [6], [7]. DRL methods have proven effective in addressing a wide range of resource allocation problems [8]. DRL has been developed as a powerful learning framework that can handle the large state space and real-time state transitions in a network environment without any prior knowledge [9], which is well-suited to dealing with complex environments that have a wide range of potential outcomes. By using DRL, it is possible to automate the Management and Orchestration (MANO) of network, enabling it to adapt to the changing network conditions (traffic patterns, e.g.) and requests in real-time.

C. Safe Reinforcement Learning

In safety-critical domains like telecommunications, traditional RL methods can inadvertently lead to hazardous actions or unacceptable outcomes. Safe RL addresses these concerns by enforcing safety constraints throughout the learning process. One effective approach integrates safety considerations into the RL objective.

Safe RL is often modeled as a Constrained Markov Decision Process (CMDP) problem [10], which requires to maximize the reward while satisfying the safety constraints or minimizing the cost. CMDP extends Markov Decision Process (MDP) by introducing a set C of cost functions $C_i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. Then the expected discounted cost-return can be defined in terms of cost function:

$$J^{C_i}(\pi) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t C_i(s_t, a_t, s_{t+1})]$$

So the feasible set of stationary policies for a CMDP is denoted as: $\Pi_C = \{\pi \in \Pi : \ \forall\ i,\ J^{C_i}(\pi) \leq d_i\}$. Consider a constrained RL problem where the agent's objective is to maximize the expected cumulative reward subject to safety constraints. The optimization problem is formulated as:

$$\max_{\pi} \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)] \quad \text{s.t.} \quad \pi \in \Pi_C$$

So the optimal policy is $\pi^* = \arg\max_{\pi_{\boldsymbol{\theta}} \in \Pi_C} J^R(\pi_{\boldsymbol{\theta}})$. Incorporating Safe RL methods empowers agents to learn policies that balance the optimization of cumulative rewards with the adherence to safety constraints.

III. State of the Art

Many Safe RL methods are built for continuous action space, used in the field of automatic vehicles and robot control. But the Core Network autoscaling typically has a discrete action space, where actions are scaling decisions. PPO is the state-of-the-art DRL algorithm for many RL tasks, and has potential to serve as a basis framework for safe methods.

A common approach to solve a CMDP problem is the Lagrangian approach [11], which is also known as the primal-dual solution. The constrained optimization problem is transformed into an unconstrained one by adjusting the objective function. This adjustment involves adding terms that consider the violations of constraints, and these terms are weighted by Lagrange multipliers associated with the constraints. Subsequently, the Lagrange multipliers are updated within the dual problem to ensure that the constraints are satisfied. Reward Constrained Policy Optimization (RCPO) [12] is a multi-timescale Lagrangian approach, in which the actor-critic policy optimization and Lagrangian multiplier have different updating timescales. The policy update is performed on a faster timescale than that of the Lagrangian penalty coefficient.

Lyapunov function is used in classic control theory. Chow et al. apply Lyapunov safe constraints function to the CMDP

problem to ensure safety and learning stability in [13] and [14]. Constrained Policy Optimisation (CPO) [15] is the first policy gradient method to solve the CMDP problem with guarantees for near-constraint satisfaction at each iteration. However, CPO's computation is more complex than the Lagrangian approach, since it needs to compute the Fisher information matrix and uses the second Taylor expansion to optimize the objectives. Based on CPO, Projection-based Constrained Policy Optimisation (PCPO) [16] is an improvement work, which constructs a cost projection to optimize the policy to a feasible region to guarantee safety. PCPO provides lower bounds on reward improvement and upper bounds on constraint violation, and displays better performance than CPO on some tasks.

However, the second-order proximal optimization is also used in PCPO, which leads to a more expensive computation cost than the first-order optimization. First Order Constrained Optimization in Policy Space (FOCOPS) method [17] has an approximate upper bound for worst-case constraint violation throughout training and only uses the first-order optimization. It is therefore simple to implement, and outperforms CPO on a set of constrained robotics locomotion tasks. Although FOCOPS is easy to implement and shows better sample efficiency, it still needs to solve the problem of unstable saddle points and unsafe actions during training [18]. Constrained Update Projection (CUP) method [19] also solves the constrained policy optimization by policy improvement and projection, providing a non-convex implementation via only first-order optimizers. In papers [20] and [21], the penalty function is designed for safety constraints and the first-order policy optimization method is provided based on it.

It is noted that there are less off-policy methods about Safe RL, especially for the Q-learning methods. In RL, techniques for selecting actions during the learning phase are called exploration/exploitation strategies, and most exploration methods have a random and greedy exploratory component, which are blind to the risk of actions [22]. Kalweit et al. [23] develop an off-policy and constrained Q learning method for autonomous driving in simulation environments. Worst-Case Soft Actor Critic(WSAC) [24] is the extension of an off-policy method called the Soft Actor Critic algorithm(SAC) [25] with a safety critic to achieve risk control. WSAC could optimize policies on the premise that its worst-case performance satisfies the constraints. Constrained Variational Policy Optimization(CVPO) [26] is an Expectation-Maximization approach to naturally incorporate constraints during the policy learning, and its performance is validated on continuous robotic tasks. There is a small number of research not focused on CMDP, but that try to introduce other MDP Formulations. In paper [27], Sun et al. address Safe RL problems under the framework of Early Terminated MDP. Sauté RL [28] and Simmer RL [29] are state augmentation approaches, which formulate Safety Augmented MDP, where the safety constraints are eliminated by augmenting the state-space with a safety state and reshaping the objective. Based on these reviews, in our pursuit to tackle the safety problem for Core Network autoscaling we turn our attention to on-policy methods, specifically PPO, described thereafter, which is the state-of-the-art on-policy method.

A. Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) [30] is a robust and widely recognized on-policy RL algorithm. PPO is distinguished by its effectiveness in addressing complex decision-making scenarios, and suitable for both continuous and discrete action space, making it an ideal candidate for our task. PPO is inspired from another policy optimization method which is called the Trust Region Policy Optimization (TRPO) [31]. PPO has most of the benefits of TRPO, and it's a first-order optimization method which is simpler to implement.

Firstly, PPO requires to compute a probability ratio $r(\theta)$ between the new policy and the old policy, which measures the difference between two policies: $r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}$. With clipped objective, we construct a new objective function to clip the estimated advantage function if the new policy is far away from the old one.The new objective function becomes:

$$J^{\mathrm{CL}}(\theta) = \mathbb{E}[\min(r(\theta)\hat{A}_\pi(s,a), \mathrm{cl}(r(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_\pi(s,a))]$$

where ε represents a small value that approximately indicates the allowable distance between the new and the old policy, $\hat{A}$ represents using Generalized Advantage Estimation(GAE) [32] to replace the original advantage function, and cl is a function depending on whether the advantage is positive or negative to clip the r and avoid moving it outside of the interval $(1-\varepsilon, 1+\varepsilon)$. In addition to the surrogate loss functions discussed above, PPO contains J^{CL} and two other losses in the objective function:

$$J(\theta) = \mathbb{E}_{\pi_\theta}[J^{\mathrm{CL}}(\theta) - c_1(\hat{V}(s_t) - R_t)^2 + c_2 H(\pi_\theta(\cdot)|s_t)]$$

where $c1$ is the value function coefficient and $c2$ is the entropy coefficient. The entropy coefficient is multiplied by the maximum possible entropy and added to loss, which helps prevent premature convergence of one action probability dominating the policy and preventing exploration.

B. From PPO to its safe versions

To address safety concerns within autoscaling problem and compare with the performance of PPO, this section introduces some Safe RL methods based on PPO.

1) PPO Lagrangian (PPO-Lag) integrates safety constraints into the agent's objective, maximizing the rewards within safe exploration. Lagrangian method is relatively simple to implement and highly scalable to different DRL algorithms. Considering both the objective and the Kullback–Leibler divergence (KL-divergence) constraints in PPO, the final optimization goals can be as follows:

$$\pi_{k+1} = \arg\max_{\pi \in \Pi_{\boldsymbol{\theta}}} J^R(\pi) \quad \text{s.t.} \quad J^{\mathcal{C}}(\pi) \leq d, KL(\pi, \pi_k) \leq \delta$$

Here, $J^{\mathcal{C}}(\pi)$ represents a safety violation constraint. The Lagrangian is introduced by augmenting the objective with a penalty term, turning the CMDP into an equivalent unconstrained problem:

$$\min_{\lambda \geq 0} \max_{\theta} \left[J^R(\pi) - \lambda J^C(\pi)\right]$$

λ is the Lagrange multiplier controlling the trade-off between reward maximization and constraint satisfaction. The Lagrangian method is a two-step process. First, solve the unconstrained problem to find a feasible solution. Then, increase the penalty coefficient until the constraint is satisfied.

- Policy update: The surrogate function in PPO is changed to satisfy the Lagrange method:

$$\max_{\pi \in \Pi_\theta} [J^R(\pi) - \lambda J^C(\pi)] \quad \text{s.t.} \quad KL(\pi, \pi_k) \leq \delta$$

- Lagrange multiplier update: After several times of the policy updates, an update on the Lagrange multiplier is performed:

$$\min_{\lambda} (1-\lambda)[J^R(\pi) - \lambda J^C(\pi)] \quad \text{s.t.} \quad \lambda \geq 0$$

  In the actual calculation process, the multiplier update on the k^{th} update is often written as:

$$\lambda_{k+1} = \max\left(\lambda_k + \eta_\lambda \left(J^C(\pi) - d\right), 0\right)$$

  where η is the learning rate of λ.

The Lagrange algorithms are sensitive to hyperparameters selection. If the initial value or the learning rate of λ is too large, the agent may suffer from a low reward, otherwise it may violate the safety constraints.

2) PPO PID-Controlled Lagrangian (PPO-PIDLag) is another variant of the Lagrange algorithm, where the update of the λ is based on the classic PID controller. But it is also known as a parameter-sensitive controller, requires tuning the control parameters (K_p, K_i and K_d) for different tasks.

3) Interior-point Policy Optimization (IPO) is a first-order policy optimization method [20] in which the logarithmic barrier functions are introduced to augment the objective and satisfy safety constraints. The empirical results on MuJoCo [33] and grid-world environments [34] have demonstrated this method's effectiveness. And IPO is also validated in the resource allocation within network slicing problem [35].

4) Penalized Proximal Policy Optimization (P3O) [21] solves the constrained policy iteration via a single minimization of an equivalent unconstrained problem. Specifically, P3O uses an effective penalty function to eliminate cost constraints and removes the trust-region constraint by the clipped surrogate objective.

IV. Experimental Setup

This section details results on two types of environment: (i) a simulated environment based on the Gym (ii) a platform based on Magma, an Open-Source vEPC framework. The pipeline for all the experiments is described in Figure 2.

A. Experiments

1) Description of the environment CoreNetTwin is a RL environment based on the Gym framework designed for the Core Network autoscaling problem[1]. It is a simulation platform that has flexible settings. Here are the setting used for the experiments:

[1]https://github.com/gfraysse/icin2024_tutorial/

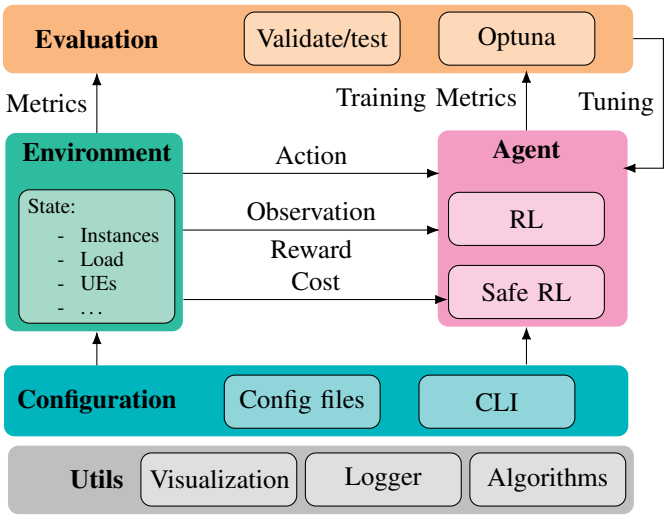


Fig. 2: Description of the experimental pipeline

- **Workload**: Figure 3 ((red curve) shows an extract of one day (out of 30) of user traffic appears in which X-axis is number of steps, Y-axis is the number of concurrent sessions. This load is from the real data of one-month VoIP calls [36].

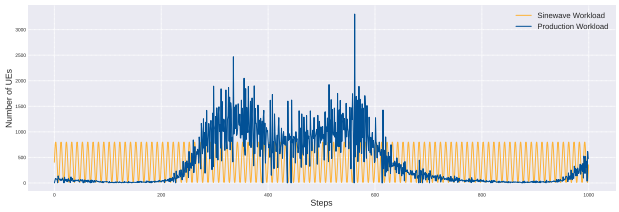


Fig. 3: Extract of the workloads, one day of production traffic used for evaluation and synthetic sine wave used for training

- **The number of available VMs**: The highest number of VMs that can be simulated in the experiment is set to 100.
- **The logic of crash simulation**: We follow the assumption that high attach rate could lead to the crash of a VM, as observed on the real-life behavior of a platform. The environment would compute the crash probability every step, the probability increases as the attach rate is higher than 3.2 based on Magma sizing recommendations[2].

2) Metrics For the autoscaling experiments, several metrics can be used to evaluate the performance and QoS:

- C**, the number of crashes**: The total number of crashes that occurred during the entire training process of the agent.
- D**, the number of sessions dropped**: The total number of sessions that failed to be set up due to insufficient resources(VMs).
- **VM usage**: How many VMs have been used during the training process.

[2]https://lf-magma.atlassian.net/wiki/download/attachments/7969523/Magma%201.7.0%20Release%20Validation.docx

- M**, Memory usage**: Each connected UE requires memory. CoreNetTwin assumes that each UE allocated 3 MB.
- R**, the attach rate**: The attach rate is defined as the rate at which new UEs are attaching to the network over a specific time period.

If sessions get dropped or a crash happens, the QoS decreases. Here, we want the C and D metrics to be close to 0. Meanwhile, we want to use less VMs to decrease the operation cost.

3) PPO baseline design We use PPO as the baseline with the following structure:

- **Input**: The observation from the environment (the number of VMs used, memory usage, the traffic load observation) is a vector formed as a 1D observation space.
- **Output**: The action can be modeled as a vector, and the action space is [-1,0,1], which means three actions: we want to scale in to decrease the resources, no action, and scale out to increase the resources.

For the NN hidden layers, a fully connected net is used with 64 units (per layer) for PPO.

4) The reward and cost functions The reward is penalized by C and D to increase sensitivity to QoS degradation. Apart from the guaranteed QoS, the platform encourages the NFs to use resources optimally around 70%. In order to reduce the number of crashes and number of D to zero, the reward function can give the penalty as big as -10. The reward function can be expressed as follows:

$$r(s,a,s') = \begin{cases} -10 & \text{if } s'_{\text{crash}} = \text{True}, \\ -D & \text{if } s'_{\text{crash}} = \text{False } \& \ D > 0, \\ 1 - |0.7 - M| & \text{if } s'_{\text{crash}} = \text{False } \& \\ & D = 0 \ \& \ M \leq 0.7, \\ -2M & \text{otherwise} \end{cases}$$

To ensure that the reward has a minimum value of -10, the reward is clipped:

$$r = \max\left(r(s,a,s'), -10\right)$$

As for the Safe RL methods, the objective is to maximize the reward function and minimize the cost function (cf. Section III-B1). The reward is thus redesigned, and the consideration of the crash is put into the cost function.

- **Reward function for Safe RL**:

$$r_{safe} = \begin{cases} -D & \text{if } D > 0, \\ 1 - |0.7 - M| & \text{if } D = 0 \ \& \ M \leq 0.7, \\ -2M & \text{otherwise} \end{cases}$$

And the reward is clamped to a minimum value of -10:

$$r = \max\left(r_{safe}, -10\right)$$

- **Cost function**: In the cost function, the cost c increases if the attach rate R is higher than 3.2, cf. Section IV-A1, and its maximum value is 10. When a crash happens, the cost is also 10.

$$\text{c} = \begin{cases} 0 & \text{if } s'_{\text{crash}} = \text{False } \& \ R \leq 3.2, \\ R - 3.2 & \text{if } s'_{\text{crash}} = \text{False } \& \ 3.2 < R \leq 13.2, \\ 10 & \text{otherwise} \end{cases}$$

It calculates the cost based on the attach rate, including the binary condition for a crash.

V. Results on a simulation environment

To compare all the algorithms fairly, the training traffic data is the same and parameters are the same except for the hyperparameters that are unique to each algorithm. Some of the important parameters are shown in Table I. The total number of the training steps is set to 48000, and the batch size is 640, which contains nearly one day's traffic data. The learning rate is for the training of the NN. Cost limit is the threshold of the cost accumulated during one batch. KL limit is one of the parameters of PPO, and since all the algorithms are based on PPO, the hyperparameters for PPO are the same for all variants.

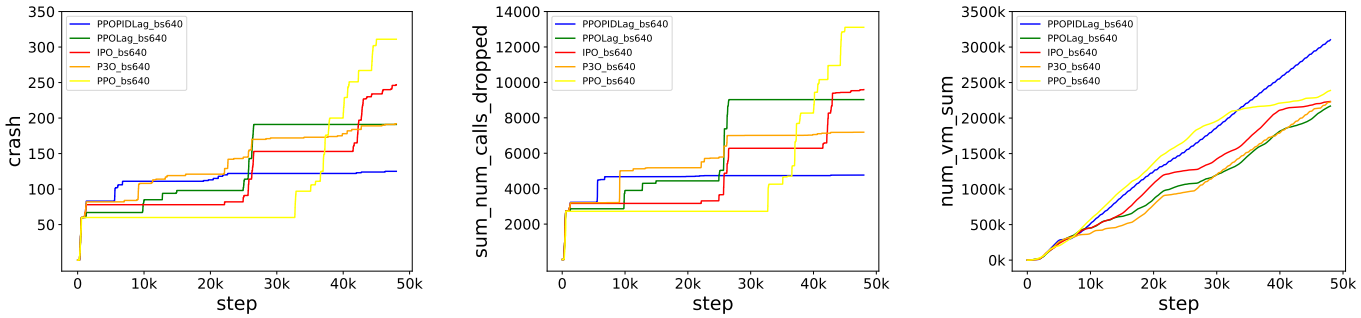
Parameters	Value
batch size	640
total number of steps	48000
learning rate	0.0001
cost limit	2
KL limit	0.2

TABLE I: Parameters settings

The results on CoreNetTwin are shown in Figure 4.

As we can see from the Figure 4a and 4b, the PPO-PIDLag algorithm (the blue curve, *PPOPIDLag-bs640*) has the best performance on the QoS metrics, it causes minimal crashes and total number of calls dropped. PPO-PIDLag also gets the convergence before 10k steps, as the crashes and calls dropped nearly stop growing after 7k steps. Conversely, PPO has the worst performance on these metrics, since it takes unsafe actions, as shown in Figure 4c, it (the yellow curve, *PPO-bs640*) keeps a lower number of VMs after 32k steps, leading to crashes and sessions dropped. On the other hand, IPO (the red curve, *IPO-bs640*) and PPO-Lag (the green curve, *PPOLag-bs640*) are also better than PPO on QoS metrics, which shows that Safe RL methods are very effective in solving our problems. Results of P3O (orange curve *P3O-bs640*) are close to those of PPO-Lag. As for the total number of VMs used, shown in Figure 4c, PPO-Lag has the lowest number of it and is the best for balancing a trade off between VM usage and QoS. As illustrated in Figure 4c, PPO uses more VMs than IPO and PPO-Lag, but still fails to satisfy the safety constraint. Comparing to other algorithms, PPO-PIDLag also uses more VMs, as we can see from the Figure 4c, although it has improved the policy to safest one. In average PPO-PIDLag uses 60 VMs, while PPO-Lag uses 45.

Hyperparameters selection was made with Optuna [37], an optimization software. It could list the importance of hyperparameters, which helped to find the most important parameters for training, e.g., different batch sizes lead to



(a) Total number of crashes (b) Total number of calls dropped (c) Total number of VMs used

Fig. 4: Evaluation of the performance of PPO, PPO-Lag, PPOPIDLag, P3O and IPO on CoreNetTwin

different performance. One potential reason behind it may be that a proper batch size is beneficial for sampling data and updating PPO policy.

VI. Results on Magma platform

A. Experimental platform based on Magma

The platform we used for running the Core Network autoscaling use-case was described in a prior work [1]. This testbed is based on Magma [38], an open-source software platform that could build a flexible and extendable EPC. The release 1.8 was used and configured as LTE rather than the 5G core to be able to compare with results from previous experiments run on release 1.5 that was not 5G-ready. Although NFs are different in the 5G network architecture, it follows the same scaling logic when using the platform. Magma packages several 3GPP NFs of EPC networks, such as MME, PGW and SGW, into a single element, called the Access Gateway (AGW) [1], so the scaling operation is directly an action on the Magma AGWs. The whole platform is deployed on VMs on the Orange Flexible Engine Cloud infrastructure.

During the training process some actions could lead some of the instances of the Access Gateway (AGW) to crash:

- If a scale-in action is performed when load is high.
- If no action is performed when the testbed is overloaded.

Since it's a real platform, there are some issues:

- **Delays**: The operations of the VMs can cause delays. Scaling in/out VM takes time, up to a few minutes.
- **Crashes**: Although data analysis was performed, it is not possible to identify all the possible reasons for them because a software can fail in many different ways when it is overloaded.

B. Experiments

1) Description of the environment

- **Workload**: The workload used for the test follows a sine wave, repeated for the duration of the experiment: In average, there are 3.33 new sessions every seconds for 180s, then waits for 180s. And the durations of each session follows a lognormal distribution, with mean = 60 and σ = 0.25. The average duration is 180s. This workload is shown in Figure 3 (blue curve).
- **The number of available VMs**: 5

2) Metrics The metrics used on this platform:

- D is the number of sessions dropped,
- C is the number of crashes caused,
- M is the memory usage in MME process,
- U is the number of connected UEs per AGW instance,
- N is the count of running AGWs.

3) Algorithm used

- **Input observation**: The set of metrics observed from the environment is a 1D observation-vector.
- **Output**: The action vector, as defined for CoreNetTwin.

In this environment, the platform also encourages the NFs to use resources(such as M and U) optimally around 70%. And the reward is penalized by D scaled by a factor of 10 to increase sensitivity to QoS degradation outside the acceptable resources usage range. Unlike CoreNetTwin , all the metric values are normalized, so the reward shaping is done using the normalized values and clipped to -1 to 1.

- **Reward function**:

$$\mathrm{r}_c = \begin{cases} 1 - |0.7 - \max(M,U)| - D & \text{if } M \leq 0.8 \\ & \& \ U \leq 0.8, \\ \max\left(-\max(M,U) - 10 \cdot D, -1\right) & \text{if } M \leq 0.8 \\ & \& \ U > 0.8, \\ -M & \text{otherwise} \end{cases}$$

One reason of the crash is that the AGW can crash when the MME overloads due to a high number of UEs trying to establish sessions, so the design of the cost function is based on the attach rate.

- **Cost function**: In the cost function, the cost c increases if the attach rate R is higher than the threshold r_{safe}, and its maximum value is 10. When a crash happens, the cost will also be 10.

$$\mathrm{c} = \begin{cases} 0 & \text{if } s'_{\text{crash}} = \text{False} \ \& \ R \leq r_{safe}, \\ R - r_{safe} & \text{if } s'_{\text{crash}} = \text{False} \ \& \\ & r_{safe} < R \leq r_{safe} + 10, \\ 10 & \text{otherwise} \end{cases}$$

4) Results The performance of the PPO-Lag model was tested on this platform. There were 6 crashes during 3k steps. It took 134 hours to run the experiment.

The detailed results are shown in Table II. The values are averaged over all steps of the experiment. PPO-Lag has the lowest average number of dropped sessions D as well as as lowest average number N of VM/AGW.

	DQN	D3QN	PPO-Lag
D	0.07	0.079	0.001
N	3.146	4.62	2.903
M	47.7	113.52	58.68
U	265.19	336.64	207.68

TABLE II: Results

It seems that PPO-Lag learns to avoid crashes while leading a trade off between QoS and operation cost.

VII. Conclusion and future work

This article introduces a Safe RL model for the Core Network autoscaling problem. The results of experiments run both on a simulation environment and a EPC platform showed that PPO-Lag has a safer behavior and leads to a more stable platform during exploration. Simulation results of other Safe RL algorithms IPO, P3O and PPO-PIDLag also confirmed safer behaviors. These results are a first step to integrate a safer behavior during the training process on a critical telco infrastructure. In future work, the Gym-based simulation platform will be improved to be a real Network Digital Twin and will be open-sourced. Since only PPO-Lag is evaluated on Magma in this work, more experiments have to be done to confirm the performance of Safe RL algorithms on a real world environment.

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*. IEEE, 2023, pp. 1–6.

[2] H. T. Nguyen, T. Van Do, and C. Rotter, "Scaling UPF Instances in 5G/6G Core With Deep Reinforcement Learning," *IEEE Access*, vol. 9, pp. 165 892–165 906, 2021.

[3] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[4] K. Arulkumaran, M. P. Deisenroth *et al.*, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[5] V. François-Lavet, P. Henderson *et al.*, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.

[6] X. Pan, Y. You *et al.*, "Virtual to real reinforcement learning for autonomous driving," *arXiv preprint arXiv:1704.03952*, 2017.

[7] J. Orr and A. Dutta, "Multi-agent deep reinforcement learning for multi-robot applications: a survey," *Sensors*, vol. 23, no. 7, p. 3625, 2023.

[8] Q. Mao, F. Hu, and Q. Hao, "Deep learning for intelligent wireless networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2595–2621, 2018.

[9] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[10] E. Altman, *Constrained Markov decision processes*. Routledge, 2021.

[11] Y. Chow, M. Ghavamzadeh *et al.*, "Risk-constrained reinforcement learning with percentile risk criteria," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6070–6120, 2017.

[12] C. Tessler, D. J. Mankowitz, and S. Mannor, "Reward constrained policy optimization," *arXiv preprint arXiv:1805.11074*, 2018.

[13] Y. Chow, O. Nachum *et al.*, "A lyapunov-based approach to safe reinforcement learning," *Advances in neural information processing systems*, vol. 31, 2018.

[14] ——, "Lyapunov-based safe policy optimization for continuous control," *arXiv preprint arXiv:1901.10031*, 2019.

[15] J. Achiam, D. Held *et al.*, "Constrained policy optimization," in *International conference on machine learning*. PMLR, 2017, pp. 22–31.

[16] T.-Y. Yang, J. Rosca *et al.*, "Projection-based constrained policy optimization," *arXiv preprint arXiv:2010.03152*, 2020.

[17] Y. Zhang, Q. Vuong, and K. Ross, "First order constrained optimization in policy space," *Advances in Neural Information Processing Systems*, vol. 33, pp. 15 338–15 349, 2020.

[18] S. Gu, L. Yang *et al.*, "A review of safe reinforcement learning: Methods, theory and applications," *arXiv preprint arXiv:2205.10330*, 2022.

[19] L. Yang, J. Ji *et al.*, "Constrained update projection approach to safe policy optimization," *Advances in Neural Information Processing Systems*, vol. 35, pp. 9111–9124, 2022.

[20] Y. Liu, J. Ding, and X. Liu, "Ipo: Interior-point policy optimization under constraints," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 04, 2020, pp. 4940–4947.

[21] L. Zhang, L. Shen *et al.*, "Penalized proximal policy optimization for safe reinforcement learning," *arXiv preprint arXiv:2205.11814*, 2022.

[22] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[23] G. Kalweit, M. Huegle *et al.*, "Deep constrained q-learning," *arXiv preprint arXiv:2003.09398*, 2020.

[24] Q. Yang, T. D. Simão *et al.*, "Wcsac: Worst-case soft actor critic for safety-constrained reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, 2021, pp. 10 639–10 646.

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

[26] Z. Liu, Z. Cen *et al.*, "Constrained variational policy optimization for safe reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 13 644–13 668.

[27] H. Sun, Z. Xu *et al.*, "Safe exploration by solving early terminated mdp," *arXiv preprint arXiv:2107.04200*, 2021.

[28] A. Sootla, A. I. Cowen-Rivers *et al.*, "Sauté rl: Almost surely safe reinforcement learning using state augmentation," in *International Conference on Machine Learning*. PMLR, 2022, pp. 20 423–20 443.

[29] A. Sootla, A. Cowen-Rivers *et al.*, "Enhancing safe exploration using safety state augmentation," *Advances in Neural Information Processing Systems*, vol. 35, pp. 34 464–34 477, 2022.

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

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

[32] J. Schulman, P. Moritz *et al.*, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[33] Y. Tassa, Y. Doron *et al.*, "Deepmind control suite," *arXiv preprint arXiv:1801.00690*, 2018.

[34] J. Leike, M. Martic *et al.*, "Ai safety gridworlds," *arXiv preprint arXiv:1711.09883*, 2017.

[35] Y. Liu, J. Ding, and X. Liu, "Resource allocation method for network slicing using constrained reinforcement learning," in *2021 IFIP Networking Conference (IFIP Networking)*. IEEE, 2021, pp. 1–3.

[36] A. Diamanti, J. Manuel Sanchez Vilchez, and S. Secci, "LSTM-based radiography for anomaly detection in softwarized infrastructures," in *International Teletraffic Congress*, IEEE. Osaka, Japan: IEEE, Sep. 2020. [Online]. Available: https://hal.science/hal-02917660

[37] T. Akiba, S. Sano *et al.*, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.

[38] "Magma," [Online]. [Online]. Available: https://magmacore.org/