Improving Policy Optimization via ε-Retrain

Luca Marzari*
University of Verona
Verona, Italy
luca.marzari@univr.it

Priya L. Donti
Massachusetts Institute of Technology
Cambridge (MA), USA
donti@mit.edu

Changliu Liu
Carnegie Mellon University
Pittsburgh (PA), USA
cliu6@andrew.cmu.edu

Enrico Marchesini
Massachusetts Institute of Technology
Cambridge (MA), USA
emarche@mit.edu

ABSTRACT

We present *ε-retrain*, an exploration strategy encouraging a behavioral preference while optimizing policies with monotonic improvement guarantees. To this end, we introduce an iterative procedure for collecting *retrain areas*—parts of the state space where an agent did not satisfy the behavioral preference. Our method switches between the typical uniform restart state distribution and the retrain areas using a decaying factor ε, allowing agents to retrain on situations where they violated the preference. We also employ formal verification of neural networks to provably quantify the degree to which agents adhere to these behavioral preferences. Experiments over hundreds of seeds across locomotion, power network, and navigation tasks show that our method yields agents that exhibit significant performance and sample efficiency improvements.

KEYWORDS

Reinforcement learning; policy gradient; formal verification

ACM Reference Format:
Luca Marzari*, Priya L. Donti, Changliu Liu, and Enrico Marchesini. 2025. Improving Policy Optimization via ε-Retrain. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 13 pages.

1 INTRODUCTION

By balancing the trade-off between *exploration* and *exploitation*, a reinforcement learning (RL) agent typically relies on a scalar reward function to learn behaviors capable of solving a task [33]. However, these functions often lead to unforeseen behaviors, making it difficult to enforce particular behaviors that we desire the system to exhibit [2]—a *behavioral preference*.

For example, consider applying a policy optimization RL method to a robot learning to reach random targets. Commonly, the agent gets a positive reward based on its distance from the goal, a penalty for collisions [35, 38], and we use a uniform restart distribution throughout the state space to randomly initialize the environment at each episode. In this setup, learning good navigation behaviors while satisfying a behavioral preference (or desiderata, interchangeably) related to safety such as *"avoid collisions"* requires many collisions around the same state. However, the uniform restart distribution naturally makes it harder for agents to experience these similar collisions over time despite being pivotal for guaranteeing monotonic policy improvement [14]. This potentially translates into a higher variance in the local estimate of the objective [31], making it hard to effectively enforce the desired behavior [6, 12].

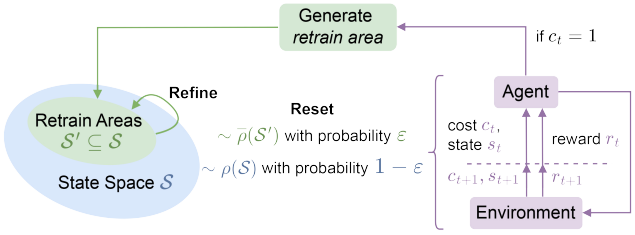


Figure 1: Explanatory overview of *ε-retrain*.

Previous policy-based approaches investigate the impact of restoring the environment to specific states to improve performance while maintaining theoretical guarantees on monotonic improvement [7, 15, 17]. A leading example is the *vine* Trust Region Policy Optimization (TRPO) algorithm [31], designed to enhance exploration and reduce the variance of gradient updates. TRPO *vine* restarts the agent in states visited by the current policy to generate additional rollouts from that state and reduce the policy update variance. The authors demonstrate how the theoretically justified procedure retains monotonic policy improvement guarantees. However, this method and, more generally, designing a poor restart state distribution has three critical downsides we address in our work.

- To the best of our knowledge, policy optimization works have not considered restarting distribution mechanisms geared towards improving specific behavioral preferences.
- Sub-optimal distributions can cause approximate methods to get stuck in local optima, potentially resulting in poor agent performance [14, 15].
- *vine* significantly hinders sample efficiency, requiring many additional rollouts for each policy update.

This paper presents *ε-retrain*, a novel exploration strategy designed to optimize policies, maintaining monotonic improvement guarantees while encouraging a behavioral preference. Our method is inspired by human learning, where consistently repeating tasks

Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Y. Vorobeychik, S. Das, A. Nowé (eds.), May 19 – 23, 2025, Detroit, Michigan, USA. © 2025 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).
* Work performed while at Carnegie Mellon University.

enhances the learning of a particular behavior. As detailed in Figure 1, we exploit an ε decay strategy to combine the uniform restart state distribution over the state space (blue), typical of RL algorithms, with a restart strategy over *retrain areas* (green). The latter uses an iterative procedure that collects and refines (i.e., creates and merges) portions of the state space where the agent violates the behavioral preference at training time. The proposed approach "retrains" the agent from these areas, improving the advantage estimation of actions violating the desired behavior, according to a probability ε (purple). A decaying schedule for ε also allows us to maintain the asymptotic convergence properties of the underlying RL algorithm and the design of retraining areas avoids sub-optimal distributions as demonstrated by our experiments. Our method also does not require additional rollouts and we prove that using mixed uniform restart distributions leads, in the worst case, to the same monotonic improvement guarantees as in Schulman et al. [31].

Wefi rst show the benefits of employing *ε-retrain* in adhering to behavioral preferences in an unconstrained setup (where we penalize the reward upon violating the preference). To this end, we evaluate *ε-retrain* on top of policy optimization methods (i.e., TRPO [31] and Proximal Policy Optimization (PPO) [32]) over hundreds of seeds and different behavioral preferences related to safety—velocity limits in locomotion tasks, preventing overloads in power grids, and collision avoidance in mobile navigation. Following our interest in safe behaviors, we also combine *ε-retrain* with the Lagrangian implementations of TRPO and PPO since they have been recently employed to enforce similar behaviors [10, 30, 34]. These Lagrangian algorithms are widely used in safe RL literature, and we use them as additional baselines for a more comprehensive evaluation. Our experiments consider diversified tasks ranging from simulated locomotion to optimizing power networks and robotic navigation, which is a commonly employed task in the RL literature [3, 19, 35, 38]. The results show that enhancing policy optimization methods with *ε-retrain* leads to significantly higher sample efficiency and better enforce the desiderata while solving the tasks. Additionally, we note that system designers typically evaluate agents empirically and can not provably quantify the degree to which they adhere to the behavioral preferences. Since our theory refers to the improvement over the main reward objective, we employ a formal verification (FV) of neural networks tool to provably quantify the rate at which the agent trained for the navigation task avoids collisions in the retrain areas.[1] Finally, the realistic environment employed in the navigation task enables the transfer of policies trained in simulation on ROS-enabled platforms. Hence, we show the effectiveness of *ε-retrain* in a realistic unsafe navigation scenario.

2 PRELIMINARIES AND RELATED WORK

We consider problems defined as Markov decision processes (MDPs), modeled as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \rho, R, \gamma)$; $\mathcal{S}$ and $\mathcal{A}$ are thefi nite sets of states and actions, respectively, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition probability distribution, $\rho : \mathcal{S} \to [0, 1]$ is the initial uniform state distribution, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, and $\gamma \in [0, 1)$ is the discount factor. In policy optimization algorithms, agents learn a parameterized stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, modeling the probability to take an action $a_t \in \mathcal{A}$ in a state $s_t \in \mathcal{S}$ at a certain step t. The goal is tofi nd the parameters that maximize the expected discounted reward $\psi(\pi) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$, where $\tau := (s_0, a_0, s_1, a_1, \dots)$ is a trajectory with $s_0 \sim \rho(s_0)$, $a_t \sim \pi(a_t|s_t)$, $s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t)$. We also define state and action value functions V_π and Q_π modeling the expected discount return starting from the state s_t (and action a_t for Q_π) and following the policy π thereafter as: $V_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \dots}[\sum_{i=0}^{\infty} \gamma^i R(s_{t+i}, a_{t+i})]$ and $Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \dots}[\sum_{i=0}^{\infty} \gamma^i R(s_{t+i}, a_{t+i})]$. Given the current state and action, we can also measure how much better or worse the agent performs compared to its expected performance—the advantage function $A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$.

To derive a bound on the policy improvement, Schulman et al. [31] also define the expected advantage of a new policy π' over the old π, and relate the expected discounted return of π' to π: $\tilde{A}(s) = \mathbb{E}_{a\sim\pi'(\cdot|s)}[A_\pi(s,a)]$, and $\psi(\pi') = \psi(\pi) + \mathbb{E}_{\tau\sim\pi'}[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t)]$. In practice, the dependency on trajectories induced by π' makes the above equation hard to optimize. To address this, the authors introduce a surrogate local approximation $L_\pi(\pi')$ to $\psi(\pi')$, using the state distribution over the current policy π rather than π':

$$\begin{aligned} L_\pi(\pi') &= \psi(\pi) + \sum_s \rho_\pi(s) \sum_a \pi'(a|s) A_\pi(s,a) \\ &= \psi(\pi) + \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t \tilde{A}(s_t)\right]. \end{aligned} \tag{1}$$

With the above intuitions, they derive an upper bound on the absolute difference between the objectives:

$$\begin{aligned} |\psi(\pi') - L_\pi(\pi')| \leq \frac{4\alpha^2\gamma k}{(1-\gamma)^2} \quad \text{with } k = \max_{s,a} |A_\pi(s,a)|, \\ \alpha = D_{KL}^{\max}(\pi, \pi') = \max_s D_{KL}(\pi(\cdot|s) \;||\; \pi'(\cdot|s)). \end{aligned} \tag{2}$$

Finally, by employing the relationship between the total variation (TV) divergence and the Kullback–Leibler (KL) divergence $D_{TV}(p||q)^2 \leq D_{KL}(p||q)$ [28], Schulman et al. [31] prove the following lower bound on the policy improvement:

$$\psi(\pi') \geq L_\pi(\pi') - C D_{KL}^{\max}(\pi, \pi'), \text{ with } C = \frac{4k\gamma}{1-\gamma^2}. \tag{3}$$

Exploration in RL. In addition to the *vine* TRPO method discussed in the previous section, a range of works investigate the idea of changing the initial state distribution [5, 24, 27]. However, these works focus on improving exploration towards achieving higher returns rather than enforcing specific desired behaviors. For example, Messikommer et al. [24] uses states from past experiences to guide the agent toward states with higher payoffs. Similarly, Ecoffet et al. [5] stores and revisits promising states to explore the environment more efficiently. In contrast, *ε-retrain*: (i) focuses on refining agent behavior by repeatedly training on states where it failed to adhere to specific preferences, which makes it more applicable in tasks where behavior consistency and safety are required; and (ii) provides a lower bound on the policy improvement for mixed restart state distributions. For this reason, we believe our method is more closely related to CMDP-related literature (over which we compare in Section 6) that is discussed in the following section.

[1] We use navigation as an explanatory task for clarity since it allows us to visualize retrain areas.

2.1 Constrained MDP

Constrained RL encourages a behavioral preference, or a safety specification such as the ones we consider in our work [29, 30, 34]. To this end, the classical MDP extends to a constrained MDP (CMDP) considering an additional set of $C := \{C_i\}_{i \in n}$ indicator cost functions and $\mathbf{l} \in \mathbb{R}^n$ hard-coded thresholds for the constraints [1]. The goal of constrained RL algorithms is to maximize the expected reward while limiting the accumulation of costs under the thresholds. To this end, policy optimization algorithms typically employ the Lagrangian to transform the problem into an unconstrained one that is easy to implement over existing algorithms [26].

Consider the case of a single constraint characterized by a cost function $C : \mathcal{S} \times \mathcal{A} \to \{0, 1\}$—define the expected cost function $\psi_C(\pi) := \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t C(s_t, a_t)\right]$, and a cost threshold l. The Lagrangian applies a differentiable penalty $\mathcal{L}_C(\lambda) = -\lambda\left(\psi_C(\pi) - l\right)$ to the policy optimization objective, where λ is the so-called *Lagrangian multiplier*. These algorithms thus take an additional gradient descent step in λ: $\nabla_\lambda \mathcal{L}_C(\lambda) = l - \psi_C(\pi)$. The multiplier is forced to be ≥ 0 as it acts as a penalty when the constraint is not satisfied (*i.e.*, λ increases) while decreasing to 0 and removing any penalty when the constraint holds. However, choosing arbitrarily small values for the threshold potentially causes a detrimental trade-off between the main task and cost objectives, ultimately leading to policies that fail to solve the problem for which they are trained. Moreover, the cost metric employed in the safety evaluation is purely empirical and does not provide any provable guarantees on the actual adherence to behavioral preferences. To address these issues, we leverage formal verification of neural networks.

2.2 Formal Verification of Neural Networks

FV is relevant to our work since it allows us to formalize a behavioral preference and provide provable guarantees on the adherence to these preferences. A reachability-based FV for neural networks tool takes as input a tuple $\mathcal{T} = \langle \mathcal{F}, \mathcal{X}, \mathcal{Y} \rangle$, where $\mathcal{F}$ is the trained policy (i.e., the neural network), and $\langle \mathcal{X}, \mathcal{Y} \rangle$ encodes a behavioral preference in terms of input-output relationships [16]. Specifically, $\mathcal{X}$ is a precondition defined on the portion of the state space we are interested in, and $\mathcal{Y}$ models the postcondition specifying the desiderata. An FV tool propagates intervals $\mathcal{X}$ through $\mathcal{F}$ and performs a layer-by-layer reachability analysis to compute the output reachable set $\mathcal{R}(\mathcal{X}, \mathcal{F})$. The tool then checks if $\mathcal{R}(\mathcal{X}, \mathcal{F}) \subseteq \mathcal{Y}$, meaning that the agent satisfies the preference for all the states in $\mathcal{X}$. Figure 2 shows a simplified overview of the verification process, which checks if (at least) one violation of the behavioral preference exists in $\mathcal{X}$. Due to over-approximation errors introduced by the propagations, FV tools iteratively split $\mathcal{X}$ into sub-domains $\mathcal{X}_i$ (the first two blocks in thefi gure) [36]. When the output reachable set $\mathcal{R}(\mathcal{X}_i, \mathcal{F})$ is not included in $\mathcal{Y}$ (the second to last block), the iterative procedure ends—the behavioral preference is violated if at least one portion of the domain $\mathcal{X}$ falls within this scenario. As a natural extension of the FV problem, recent works [21, 22, 37] propose to enumerate all the portions of $\mathcal{X}$ violating the desiderata, thus provably quantifying the rate at which agents satisfy the input-output relationships. In this work, we rely on the tool proposed by [21] to quantify the degree to which agents adhere to behavioral preferences in the explanatory robotic navigation task.

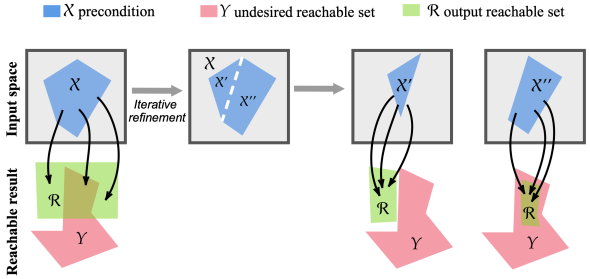


Figure 2: Overview of FV for neural networks.

3 POLICY OPTIMIZATION VIA ε-*RETRAIN*

We introduce *ε-retrain* to restart an agent from regions of the state space where it previously violated a behavioral preference. Our goal is to encourage a policy to exhibit behaviors aligned with the preference while improving performance and sample efficiency. To this end, *ε-retrain* collects retrain areas—subsets of the state space $\overline{\mathcal{S}} \subseteq \mathcal{S}$ defined using an iterative procedure that merges parts of $\mathcal{S}$ where the agent violated the preference during training. We then introduce a *mixed restart distribution*, combining the typical uniform restart distribution ρ over the entire state space $\mathcal{S}$, with another uniform restart distribution $\overline{\rho} : \overline{\mathcal{S}} \to [0, 1]$ that considers retraining areas. Crucially, such a procedure is simple to implement and can potentially be applied to any RL algorithm.

Algorithm 1 presents the pseudocode for *ε-retrain*. We start by initializing the memory buffer of the retrain areas $\overline{\mathcal{S}}$ as an empty set, and we select the starting state s_0 using the initial uniform state distribution ρ over the entire state space $\mathcal{S}$ (line 2). In the training loop, the iterative procedure for collecting and merging retrain areas begins upon each unsafe interaction which returns a positive cost signal to the agent. We use this indicator cost signal as

Algorithm 1 Template for *ε-retrain* methods

Require: *bubble* size ω for initial retrain area, *similarity* value β to merge similar areas, *decay*, *initial*, and *minimum* values for the ε scaling.

1: $\varepsilon\_decay \leftarrow (min\_\varepsilon - 1.0)/(decay \cdot (epochs \cdot steps\_per\_epoch))$
2: $\overline{\mathcal{S}} \leftarrow \emptyset$; $s_0 \leftarrow \rho(\mathcal{S})$ ▷ Initialize areas buffer and environment
3: **for** each episode **do**
4: **while** episode is not done **do**
5: Execute the training loop of the RL algorithm for each step t.
6: **if** s_t is unsafe (i.e., cost > 0) **then**
7: $r \leftarrow$ `generate_retrain_area`(s_{t-1}, ω)
8: **if** $\exists\, r' \in \overline{\mathcal{S}}$ that is β-similar to r **then**
9: $r \leftarrow$ `area_refinement`(r, r')
10: **end if**
11: $\overline{\mathcal{S}} \leftarrow \overline{\mathcal{S}} \cup r$
12: **end if**
13: **end while**
14: **if** $random(0, 1) < \varepsilon \wedge \overline{\mathcal{S}} \neq \emptyset$ **then**
15: $s_0 \leftarrow \overline{\rho}($`sample_retrain_area`$(\overline{\mathcal{S}}))$
16: $\varepsilon \leftarrow \max(\varepsilon\_decay \cdot (epoch) * steps\_per\_epoch + 1.0,\ min\_\varepsilon)$
17: **else**
18: $s_0 \leftarrow \rho(\mathcal{S})$
19: **end if**
20: **end for**

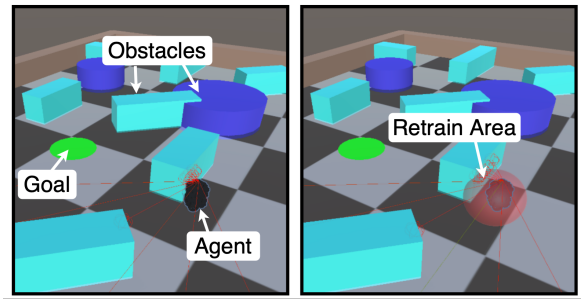


Figure 3: (left) The agent collides with an obstacle, receiving a positive cost. (right) A retrain area is created from that state.

in the safe RL literature to detect the interactions where the agent violates the desiderata [8]. Specifically, we generate a retrain area calling the `generate_retrain_area` method, which requires the previous state s_{t-1} and generate an ω-bubble (line 3-7) [23]. Broadly speaking, such an area is a portion of the space surrounding the state that led to a violation, created by encoding each feature of the agent's state as an interval withfi xed size ω (i.e., a "bubble" around the state). Figure 3 shows an explanatory example of an area for navigation, where collision avoidance is our desiderata. In this example, by leveraging commonly available information (e.g., the sensors' precision), we encode a bubble of size ω around the state that led to the collision. The idea is that retraining an agent from these similar collision-prone situations improve performance.

This bubble thus becomes a retrain area r, and our approach automatically checks for the existence of another similar retrain area r' to merge with, using the similarity threshold β provided as a parameter. If such an r' exists, the `area_refinement` method is called (line 9). If no similar retrain area is found, r is stored in the buffer $\overline{\mathcal{S}}$ that is initially empty (line 11). The refinement offers two advantages: (i) it allows us to maintain a reasonable size for $\overline{\mathcal{S}}$, and (ii) it clusters similar behavioral violations within the same retrain area, guaranteeing a uniform sampling over different violations. The new or refined area is then inserted into $\overline{\mathcal{S}}$ and can be used to retrain the agent. We refer to the next section for a complete overview of the generation and refinement methods. If there is at least one retrain area in $\overline{\mathcal{S}}$, the new initial state of the environment is either randomly sampled from the entire state space $\mathcal{S}$ with probability $1-\varepsilon$, or randomly sampled from a retrain area with probability ε (lines 14-15). *ε-retrain* employs a linear decay for the mixed restarting distribution, avoiding the problem of getting stuck in suboptimal restart distributions (lines 16-20). If the sample is from a retrain area, the environment resets to a configuration (similar to) where the agent previously violated the behavioral preference.

3.1 Generation and Refinement Processes

This section introduces the generation and refinement methods through a practical example. Wefi rst show the retrain area generation in the robotic navigation context and then the refinement process in the *HalfCheetah* locomotion task.

3.1.1 Retrain Area Generation. Suppose an agent in a navigation scenario receives a positive cost signal from the environment. This indicates a collision with an obstacle, as depicted in Figure 4(a).

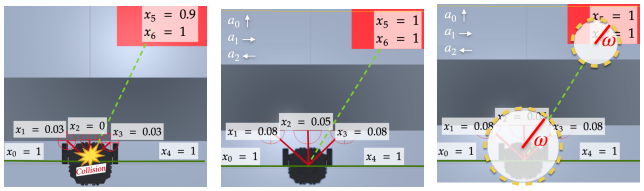


Figure 4: Retrain area generation. (a) Collision with an obstacle. (b) A previous unsafe state led to the collision. (c) ω-bubble size to initialize the retrain area. Note that the ω-bubble is the same for all the input features and is depicted in different sizes just for clarity representation purposes.

The generation procedure selects the previous state s_{t-1} that led to the collision as reported in Figure 4(b) and considers an ω-bubble around this state to generate a retrain area as in Figure 4(c). Taking $\omega = 0.05$, i.e., a small value that encodes the surroundings of an unsafe situation, we obtain one interval for each input feature as:

$$X : \{x_0 = [0.95, 1], x_1 = [0.03, 0.08], x_2 = (0, 0.05],$$
$$x_3 = [0.03, 0.08], x_4, x_5, x_6 = [0.95, 1]\}.$$

We assume the states sampled from X are undesirable—potentially risky. Therefore, when the initial state s_0 is sampled from this region X using *ε-retrain*, the agent can improve the policy by learning how to better satisfy the desired safe behavior.

3.1.2 Refinement Procedure. Once a retrain area has been created, *ε-retrain* checks whether it is possible to perform a refinement with an existing retrain area. To this end, we check if the distance between each corresponding interval in two different selected retrain areas, X and X', is less than or equal to β—a similarity threshold parameter. Formally, let $X = \{x_0, x_1, \ldots, x_n\}$ and $X' = \{x'_0, x'_1, \ldots, x'_n\}$, where each x_i and x'_i are intervals that encode possible value for each feature x_i ($\forall i \in \{0, \ldots, n\}$). We use Moore's interval algebra [25] and define the *distance* between two intervals as $[\underline{x_i}, \overline{x_i}]$ and $[\underline{x'_i}, \overline{x'_i}]$ as:

$$d([\underline{x_i}, \overline{x_i}], [\underline{x'_i}, \overline{x'_i}]) = \max(|\underline{x_i} - \underline{x'_i}|, |\overline{x_i} - \overline{x'_i}|).$$

Hence, two sets of intervals X and X', i.e., two retrain areas, are similar if and only if:[2]

$$\forall i \in \{0, \ldots, n\}, \quad d(x_i, x'_i) \leq \beta.$$

Figure 5 shows an example in a locomotion task, depicting a similar and not similar unsafe situation. This task has a desired velocity threshold for the Cheetah along the x axis, and the red ball in the center of the image indicates a violation of such a threshold. For clarity, eachfi gure uses two unsafe states (one with the original color and the other with afi xed red or green color) instead of intervals. If two sets of intervals X and X' are similar (leftfi gure), *ε-retrain* combines these areas into a new area with intervals X''. For each pair of corresponding intervals $x_i \in X$ and $x'_i \in X'$ for all $i \in \{0, \ldots, n\}$, the new interval $x'' \in X''$ is computed as:

[2]Without loss of generality, we assume the l2-norm in the features space is a meaningful distance metric. Using different metrics in different scenarios (e.g., robotic manipulation) does not impact our refinement procedure.

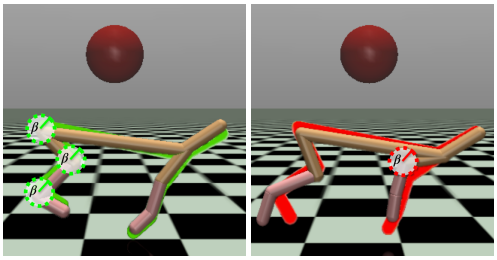


Figure 5: Left: Explanatory similar unsafe states for a subset of the input features—the two states are within distance β. Right: Explanatory different unsafe situations—at least a couple of input features have a distance greater than β.

$$x_i'' = [\min(\underline{x_i}, \underline{x_i'}), \max(\overline{x_i}, \overline{x_i'})],$$

meaning we take the minimum of the lower bounds and the maximum of the upper bounds of the corresponding intervals. Otherwise (right image), we keep the two areas separate.

4 POLICY IMPROVEMENT

In this section, we derive a bound on the monotonic policy improvement for ε-*retrain*'s mixture of uniform restart distributions. We show that the original bound on monotonic policy improvement presented by Schulman et al. [31] still holds and can be tighter. Notably, *this result motivates both the design of our method as well as its superior performance* (Section 6). For the sake of clarity, we first recall the definition of α-*coupled policies* and the related lemma introduced to derive the bound in case of a single uniform restart distribution over $\mathcal{S}$. First, Definition 1 couples two policies that behave in the same way (i.e., given a state, they pick the same action) with probability $\geq 1-\alpha$, and Lemma 1 bounds the gap between policy advantages satisfying such policies.

Definition 1 (α-coupled policiesS chulman et al. [31]). *We say that π and π' are two α-coupled policies if $\forall s \in \mathcal{S}$, we can define a joint distribution $(a,a')|s$ such that $P(a \neq a'|s) \leq \alpha$.*

Lemma 1 (Schulman et al. [31]). *Given two α-coupled policies, π and π', we have that: $\left|\mathbb{E}_{s_t\sim\pi'}[\tilde{A}(s_t)] - \mathbb{E}_{s_t\sim\pi}[\tilde{A}(s_t)]\right| \leq 4\alpha(1-(1-\alpha)^t)\max_{s,a}|A_\pi(s,a)|$. It follows that: $|\psi(\pi')-L_\pi(\pi')| \leq \frac{4\alpha^2\gamma k}{(1-\gamma)^2}$ with $k = \max_{s,a}|A_\pi(s,a)|$.*

We extend Lemma 1 to the mixture of restarting distributions used by ε-*retrain* (right side of Equation 4, where $s_0 \sim \overline{\rho}$). To this end, we define the expected discounted return of a new policy π' over the current π under the ε mixture of restart policies as:

$$\begin{aligned}\overline{\psi}(\pi') = (1-\varepsilon)\left[\psi(\pi) + \mathbb{E}_{\substack{\tau\sim\pi'\\ s_0\sim\rho}}\left[\sum_{t=0}^{\infty}\gamma^t\tilde{A}(s_t)\right]\right] \\ + (\varepsilon)\left[\psi(\pi) + \mathbb{E}_{\substack{\tau\sim\pi'\\ s_0\sim\overline{\rho}}}\left[\sum_{t=0}^{\infty}\gamma^t\tilde{A}(s_t)\right]\right].\end{aligned} \tag{4}$$

Hence, for the surrogate loss (see Equation 1), we compute the following local approximation:

$$\begin{aligned}\overline{L}_\pi(\pi') = (1-\varepsilon)\left[\psi(\pi) + \mathbb{E}_{\substack{\tau\sim\pi\\ s_0\sim\rho}}\left[\sum_{t=0}^{\infty}\gamma^t\tilde{A}(s_t)\right]\right] \\ + (\varepsilon)\left[\psi(\pi) + \mathbb{E}_{\substack{\tau\sim\pi\\ s_0\sim\overline{\rho}}}\left[\sum_{t=0}^{\infty}\gamma^t\tilde{A}(s_t)\right]\right].\end{aligned} \tag{5}$$

We then extend Lemma 1 to provide an upper bound on the distance between $\overline{\psi}(\pi')$ and $\overline{L}_\pi(\pi')$ under ε-*retrain*.

Lemma2. *Considering a mixed restart distribution over ρ and $\overline{\rho}$ using ε-retrain, it holds that*

$$|\overline{\psi}(\pi') - \overline{L}_\pi(\pi')| \leq \frac{4\alpha^2\gamma}{(1-\gamma)^2}\left[k(1-\varepsilon)+k'\varepsilon\right] \leq \frac{4\alpha^2\gamma k}{(1-\gamma)^2},$$

$$\text{with } k' = \max_{\substack{s_0\sim\overline{\rho}\\ \tau\sim\pi'}}|A_{\pi'}(s,a)| \leq \ k = \max_{\substack{s_0\sim\rho\\ \tau\sim\pi'}}|A_{\pi'}(s,a)|,\ \varepsilon \in [0,1].$$

Proof. Full proof of Lemma 2 is presented in Appendix 8.[3] □

Finally, by exploiting the relationship between the total variation divergence and the KL divergence [28], we derive the following corollary on the monotonic improvement guarantee under ε-*retrain*-based methods.

Corollary1. *Let ρ and $\overline{\rho}$ be two different restart distributions. Combining ρ and $\overline{\rho}$ during the training as in ε-retrain, and by setting $\alpha = D_{KL}^{max}(\pi,\pi')$, the bound on the monotonic improvement for the policy update $\overline{\psi}(\pi') \geq \overline{L}_\pi(\pi') - CD_{KL}^{max}(\pi,\pi')$ with $C = \frac{4k\gamma}{1-\gamma^2}$ is still guaranteed.*

The result naturally follows from Lemma 2 and the original derivation of Schulman et al. [31].[4]

5 LIMITATIONS

We identify the three following limitations in our work:

- Our algorithm requires a simulator to train the agent and the possibility of resetting the system to specific states. We believe this requirement is reasonable in the RL literature.
- In ε-*retrain* , we assume having access to an additional indicator cost signal from the environment. Such a signal is widely adopted in the safe RL literature, where system designers assume having access to a cost function that deems a state-action pair as safe or unsafe [1, 11, 29].
- We assume the area surrounding a collision state is also prone to violations of the desiderata. When such an assumption does not hold (e.g., in highly non-linear systems such as power grids), we use the exact feature values to determine a retrain area instead of intervals of size ω.

6 EXPERIMENTS

We *present a comprehensive evaluation of ε-retrain applied to TRPO* [31] that approximates the policy optimization theory of Section 3, *and PPO* [32] which relaxes the computational demands of TRPO.[5]

[3] All the appendices are available at this link.

[4] We note the theoretically justified procedure motivating the policy improvement bound [31] does not hold for most practical deep RL policy optimization algorithms.

[5] We also tested *vine* TRPO, which achieved comparable results with lower sample efficiency than TRPO. For this reason, our evaluation considers the TRPO algorithm.

We refer to these methods as ε-TRPO and ε-PPO. *Additionally, we investigate the impact of the proposed approach on top and against safe RL baselines*, using the Lagrangian method with TRPO and PPO and modeling behavioral preferences as constraints. The resulting algorithms are named TRPOLagr, PPOLagr, ε-TRPOLagr, and ε-PPOLagr. Our experiments address the following questions:

- Does ε-*retrain* allow agents to better adhere to behavioral preferences while solving the task in both an unconstrained (where we penalize the reward upon violating the preference) and constrained formalization?
- How does ε-*retrain* impact existing CMDP-based methods aimed at satisfying these preferences?
- How often do agents satisfy the behaviors provably and empirically?

To answer these questions, we begin our experiments using two known safety-oriented tasks, "SafetyHopperVelocity-v1, and "SafetyHalfCheetahVelocity-v1", from the Safety-Gymnasium benchmark [10]. To evaluate our method in a variety of setups and different behavioral desiderata, we also employ two practical scenarios based on an active network management task for a power system [9], and navigation for a mobile robot [4, 18]. Figure 6 shows these tasks. For simplicity, we will refer to them as *Hopper*, *HalfCheetah* (Cheetah), *Active Network Management* (ANM), and *Navigation*, respectively. In the following, we briefly describe the tasks, referring to Appendix 9 for a more exhaustive description.

- Hopper, Cheetah: The robots have to learn how to run forward by exerting torques on the joints and observing the body parts' angles and velocities (for a total of 12 and 18 input features). The actions control the torques applied to the (3 and 6) joints of the robot.
- ANM: The agent has to reschedule the power generation of different renewable and fossil generators, to satisfy the energy demand of three loads connected to the power grid. The agent observes the state of the power network through 18 features (i.e., active and reactive power injections, charge levels, and maximum productions) and controls power injections and curtailments using 6 continuous actions.
- Navigation: A mobile robot has to control its motor velocities to reach goals that randomly spawn in an obstacle-occluded environment without having a map. The agent observes the relative position of the goal and sparse lidar values sampled at afi xed angle (for a total of 22 features) and controls linear and angular velocity using 2 continuous actions.

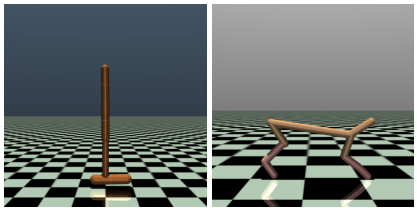
6.1 Implementation Details

Data collection is performed on Xeon E5-2650 CPU nodes with 64GB of RAM, using existing implementations for PPO, TRPO, and their Lagrangian version, based on the omnisafe library [11]. Complete hyperparameters are in Appendix 12. We report the average return, cost, and standard error as shaded regions over 50 independent runs per method. Figures 7 and 12 (the latter reported in the supplementary) show the average return in thefi rst row and the average cost in the second row, where each column represents a different task. Notably, *we are seeking agents that achieve a lower cost, which indicates they better adhere to the desired behavioral preferences while also solving the task*. Our claims on the performance improvement of ε-*retrain* are supported by 1600 training runs, which significantly surpasses the typical 3-10 runs per method used in previous policy optimization works [31, 32]. We note that due to employing a small penalty in the reward function to encourage specific behavioral preference, our results are not directly comparable to the published baselines [31, 32]. For a fair comparison, wefi rst collect the baseline with this new setting and then compare the performance with our approach. Considering the computational resources used for our extensive evaluation, Appendix 11 addresses the environmental impact of our experiments.

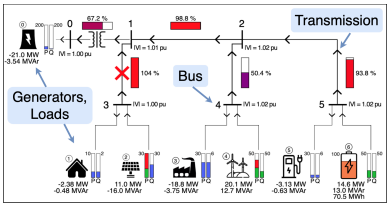
6.2 Empirical Evaluation

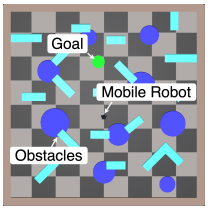
Performance of ε-TRPO and ε-PPO. Figure 7 shows that TRPO and PPO enhanced with our ε-*retrain* improve sample efficiency and allow agents to better adhere to the behavioral preferences.

In Hopper and HalfCheetah, TRPO and PPO achieve substantially higher returns than their ε-*retrain* version; a result that could be easily misunderstood. In fact, this is related to the nature of the task, where the reward is directly proportional to the agents' velocities. For this reason, an agent that violates the behavioral preference *"limit velocity under a threshold"*, achieves higher returns. This is clearly shown in thefi rst two columns of Figure 7, where ε-TRPO and ε-PPO resulted in notably lower cost compared to the baselines, indicating they lead the agents towards adhering to the velocity limit significantly more often than TRPO and PPO. Similar results are achieved in the ANM task where ε-TRPO and ε-PPO are notably safer and more sample efficient (see Pareto frontiers for convergence results in Figure 11 in the supplementary) than the baseline counterparts. The benefits of ε-*retrain* are also confirmed in the navigation task, where violating the safety desiderata *"avoid collisions"* leads to more collisions. Ultimately, achieving a higher cost (i.e., more collisions) hinders the navigation performance of the agent and leads to lower returns. Specifically, TRPO and ε-TRPO converge to the same average cost. However, the higher sample efficiency of the latter through the training, in terms of learning collision avoidance behaviors more quickly, allows ε-TRPO to learn



(a) Hopper (b) HalfCheetah





(c) Active Network Management (d) Navigation

Figure 6: Environments employed in our experiments.

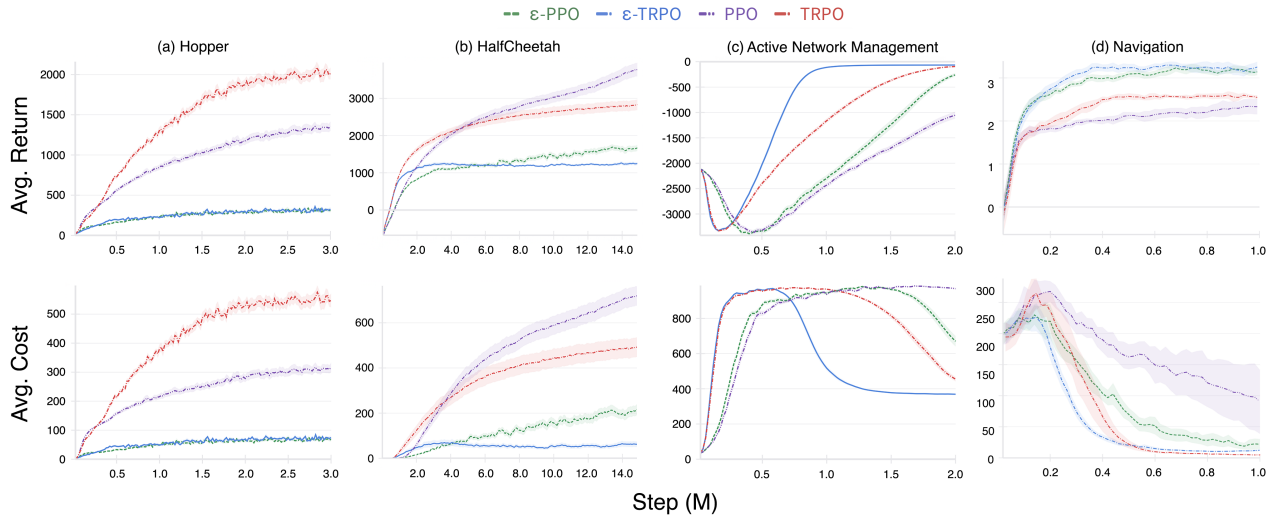


Figure 7: Comparison of ε-PPO, ε-TRPO, PPO and TRPO.

Table 1: Average fraction of the training steps where agents violate the constraints (lower is better).

	Hopper	Cheetah	ANM	Navigation
PPOLagr	0.57	0.33	**0.44**	0.46
ε-PPOLagr	**0.04**	**0**	0.59	**0.44**
TRPOLagr	0.51	0.17	**0.58**	0.64
ε-TRPOLagr	**0.25**	**0**	0.79	**0.56**

better navigation behaviors, outperforming TRPO in terms of average return. Moreover, ε-PPO significantly outperforms PPO both in terms of average cost and return.

Performance of ε-TRPOLagr and ε-PPOLagr. During training for the Lagrangian algorithms, both ε-TRPOLagr and ε-PPOLagr drastically reduce the amount of constraint violations in the Hopper and HalfCheetah velocity environments (complete learning curves are reported in Appendix 10). We specify the fraction of training steps where agents violate their constraint in Table 1. However, at convergence, all the approaches satisfy the imposed thresholds. Figure 8 shows the Pareto frontier reporting on the y-axis the average reward and on the x-axis the average cost at convergence. These results lead to some interesting considerations based on the setup of interest. In safety-critical contexts where it is crucial to satisfy constraints at training time, *ε-retrain* showed significant empirical benefits. On the other hand, in non-critical contexts where performance at convergence is the main evaluation metric, the naive Lagrangian methods have superior return performance. Intuitively, this relates to the fact that Lagrangian methods often violate the constraints at training time, allowing agents to explore more and thus learn higher-performing behaviors. In the more complex, realistic scenarios, our empirical analysis leads to different considerations. Specifically, in navigation, *ε-retrain*-based methods and the Lagrangian baselines achieve comparable results in terms of cost (i.e., constraint satisfaction). However, retraining agents in areas that are collision-prone allowed them to learn policies with better navigation skills and higher performance. In the ANM task, retraining an agent during grid instability increases the frequency of constraint violations compared to the baseline. Nonetheless, our approach helps agents learn to manage the grid effectively over time in contrast to Lagrangian baselines, which in the end, fail to solve the problem efficiently.

6.3 Provably Verifying Navigation Behaviors

To further assess the benefit that *ε-retrain* has over the behavioral preferences, we formally verify the policies trained for the navigation task. We consider this problem as an explanatory task for

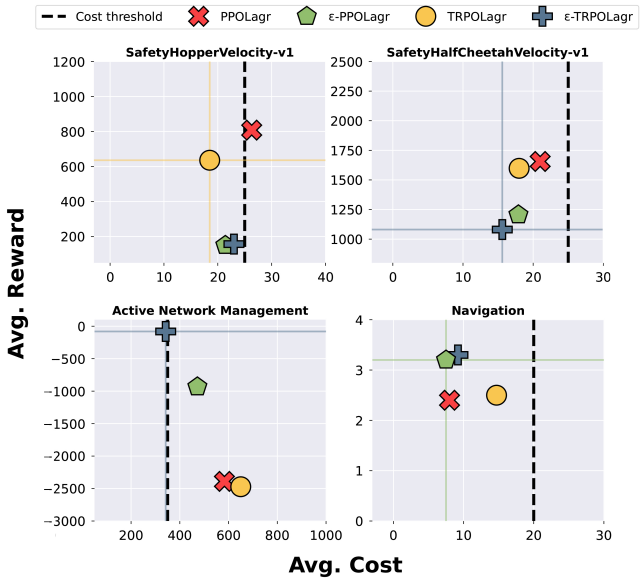


Figure 8: Pareto frontier of reward versus cost for ε-PPOLagr, ε-TRPOLagr, PPOLagr and TRPOLagr at convergence.

clarity since it allows us to easily visualize the retrain areas generated for *"collision avoidance"* on top of the environment.

Figure 9, shows a kernel density estimation map of the retrain areas distribution at the beginning (left) andfi nal stages (right) of the training for the ε-TRPO agent. Here we can notice how the agent successfully learns to navigate the environment over time since the retrain areas are more equally distributed through the entire scenario. Using a recent verification tool [21], we aim to quantify the probability that a navigation policy violates the collision avoidance preference. To this end, we consider three representative retrain areas collected while training the different ε-*retrain*-based algorithms (depicted as red dots in Figure 9). For each area, we encode the input-output relationship required by the tool (see Section 2.2), considering the retrain area as the precondition, and the minimum linear and angular velocities that would cause a collision as the postcondition. Broadly speaking, the FV tool checks where the trained policies do not exceed such minimum velocities (i.e., they do not collide), and returns the portion of each retrain area for which the given policy violates the postcondition (i.e., the probability of colliding in that area). Table 2 reports the probability that policies at convergence collide in the chosen retrain areas, averaged over all the runs. This additional FV-based analysis shows that ε-*retrain* algorithms better adhere to the behavioral preference, further confirming our intuitions and the merits of our approach.

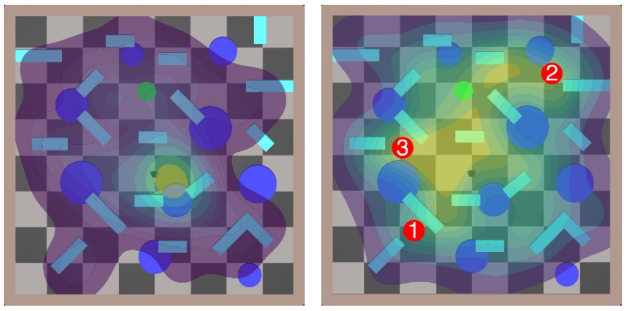


Figure 9: Density map of the retrain areas collected in the first and last training epochs; yellow indicates higher density.

Table 2: Average behavioral violations percentage for policies trained with TRPO, PPO, PPOLagr, TRPOLagr, and their ε-*retrain* version (ours).

	Retrain areas (1, 2, 3)		
ε-PPO	**0.007%**	**0.011%**	**0.22%**
PPO	0.012%	0.017%	0.59%
ε-TRPO	**0.014%**	**0.67%**	1%
TRPO	0.015%	0.69%	**0.8%**
ε-PPOLagr	**0.006%**	**0%**	**0.012%**
PPOLagr	0.013%	0.05%	0.1%
ε-TRPOLagr	0.0004%	**0.007%**	**0.46%**
TRPOLagr	**0.00005%**	0.012%	0.58%

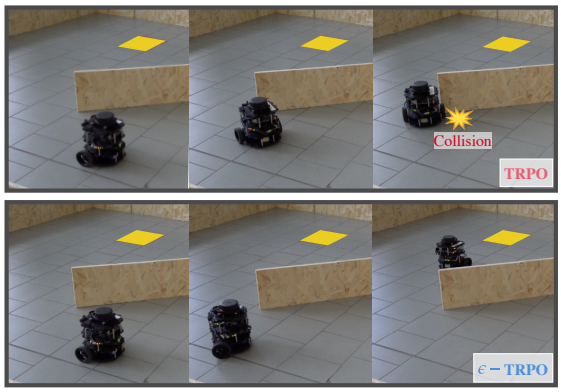


Figure 10: Real-world experiments comparing ε-TRPO and TRPO in corner-case scenarios.

6.4 Real experiments

To conclude our comprehensive evaluation, we perform an additional evaluation in realistic unsafe mapless navigation settings. Due to the similar performance at convergence for TRPO and ε-TRPO, we report the evaluation for these two approaches. Specifically, we compare ε-TRPO and TRPO in scenarios where the agent has either all or only partially occluded LiDAR information. We hypothesize that if the agent is not exposed to multiple unsafe situations during the training, i.e., without an ε-*retrain* strategy, it is less likely to select a longer but safer trajectory, and eventually, the agent will prefer a straight trajectory leading to a collision. To this end, the Unity framework [13] we used to create the navigation task allows us to transfer the policies trained in simulation on ROS-enabled platforms such as our Turtlebot3 (Fig. 10). Hence, we test several corner-case situations, comparing the safer (from the formal verification results) trained agent at convergence for both ε-TRPO and TRPO. Realistic experimental results confirm our hypothesis, showing the benefit of retraining the agent in specific regions of the state space deemed unsafe.

7 DISCUSSION

This paper presented ε-*retrain*, a novel exploration strategy with monotonic improvement guarantees that optimizes policies while encouraging specific behavioral preferences. Our approach aims at retraining an RL agent from *retrain areas* where it violated a desired behavioral preference at training time. Our empirical and formal evaluation over hundreds of seeds considering various tasks and behavioral preferences, demonstrated the effectiveness in terms of higher sample efficiency and superior performance of ε-*retrain* when integrated with existing policy optimization methods. Real-world experiments confirmed the benefit of the proposed approach in realistic setups.

ACKNOWLEDGMENTS

This work was partly supported by mobility grants for non-EU destinations at the University of Verona's Doctoral School, the AI2050 program at Schmidt Sciences (Grant G-24-66236), and the MIT Climate Nucleus Fast Forward Faculty Fund Grant Program.

REFERENCES

[1] Eitan Altman. 1999. Constrained Markov Decision Processes. In *CRC Press*.

[2] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul F. Christiano, John Schulman, and Dan Mané. 2016. Concrete Problems in AI Safety. *arXiv preprint arXiv:1606.06565* (2016).

[3] Ayhan Alp Aydeniz, Enrico Marchesini, Christopher Amato, and Kagan Tumer. 2024. Entropy Seeking Constrained Multiagent Reinforcement Learning. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*. 2141–2143.

[4] Alberto Castellini, Enrico Marchesini, Giulio Mazzi, and Alessandro Farinelli. 2020. Explaining the Influence of Prior Knowledge on POMCP Policies. In *Multi-Agent Systems and Agreement Technologies*. 261–276.

[5] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. 2021. First return, then explore. *Nature* 590 (2021), 580–586.

[6] Benjamin Eysenbach, Shixiang Gu, Julian Ibarz, and Sergey Levine. 2018. Leave no Trace: Learning to Reset for Safe and Autonomous Reinforcement Learning. In *International Conference on Learning Representations*.

[7] Victor Gabillon, Mohammad Ghavamzadeh, and Bruno Scherrer. 2013. Approximate dynamic programmingfi nally performs well in the game of Tetris. *Advances in neural information processing systems* 26 (2013).

[8] Javier Garcıa and Fernando Fernández. 2015. A comprehensive survey on safe reinforcement learning. In *JMLR*.

[9] Robin Henry and Damien Ernst. 2021. Gym-ANM: Reinforcement learning environments for active network management tasks in electricity distribution systems. *Energy and AI* 5 (2021), 100092.

[10] Jiaming Ji, Borong Zhang, Jiayi Zhou, Xuehai Pan, Weidong Huang, Ruiyang Sun, Yiran Geng, Yifan Zhong, Josef Dai, and Yaodong Yang. 2024. Safety Gymnasium: A Unified Safe Reinforcement Learning Benchmark. *Advances in Neural Information Processing Systems* 36 (2024).

[11] Jiaming Ji, Jiayi Zhou, Juntao Dai Borong Zhang, Ruiyang Sun Xuehai Pan, Weidong Huang, Yiran Geng, Mickel Liu, and Yaodong Yang. 2023. OmniSafe: An Infrastructure for Accelerating Safe Reinforcement Learning Research. *arXiv preprint arXiv:2305.09304* (2023).

[12] Yiding Jiang, J Zico Kolter, and Roberta Raileanu. 2023. On the Importance of Exploration for Generalization in Reinforcement Learning. In *Thirty-seventh Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=y5duN2j9s6

[13] Arthur Juliani, Vincent-Pierre Berges, Esh Vckay, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. 2018. Unity: A Platform for Intelligent Agents. In *CoRR*.

[14] Sham Kakade and John Langford. 2002. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*. 267–274.

[15] Michail G Lagoudakis and Ronald Parr. 2003. Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 424–431.

[16] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, Mykel J Kochenderfer, et al. 2021. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization* 4, 3-4 (2021), 244–404.

[17] Enrico Marchesini and Christopher Amato. 2023. Improving Deep Policy Gradients with Value Function Search. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=6qZC7pfenQm

[18] Enrico Marchesini and Alessandro Farinelli. 2021. Centralizing State-Values in Dueling Networks for Multi-Robot Reinforcement Learning Mapless Navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 4583–4588. https://doi.org/10.1109/IROS51168.2021.9636349

[19] Enrico Marchesini and Alessandro Farinelli. 2022. Enhancing Deep Reinforcement Learning Approaches for Multi-Robot Navigation via Single-Robot Evolutionary Policy Search. In *International Conference on Robotics and Automation (ICRA)*. 5525–5531. https://doi.org/10.1109/ICRA46639.2022.9812341

[20] Enrico Marchesini, Luca Marzari, Alessandro Farinelli, and Christopher Amato. 2023. Safe Deep Reinforcement Learning by Verifying Task-Level Properties. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*. 1466–1475.

[21] Luca Marzari, Davide Corsi, Ferdinando Cicalese, and Alessandro Farinelli. 2023. The #DNN-Verification Problem: Counting Unsafe Inputs for Deep Neural Networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*. 217–224.

[22] Luca Marzari, Davide Corsi, Enrico Marchesini, Farinelli Alessandro, and Ferdinando Cicalese. 2024. Enumerating Safe Regions in Deep Neural Networks with Provable Probabilistic Guarantees. *Proceedings of the AAAI Conference on Artificial Intelligence* (2024), 21387–21394.

[23] Luca Marzari, Enrico Marchesini, and Alessandro Farinelli. 2023. Online Safety Property Collection and Refinement for Safe Deep Reinforcement Learning in Mapless Navigation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 7133–7139.

[24] Nico Messikommer, Yunlong Song, and Davide Scaramuzza. 2024. Contrastive Initial State Buffer for Reinforcement Learning. *2024 IEEE International Conference on Robotics and Automation (ICRA)* (2024).

[25] Ramon E Moore. 1962. *Interval arithmetic and automatic error analysis in digital computing*. Technical Report. Stanford Univ Calif Applied Mathematics And Statistics Labs.

[26] J. Nocedal and S. Wright. 2006. *Numerical Optimization* (2 ed.). Springer.

[27] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. 2018. DeepMimic: Example-guided Deep Reinforcement Learning of Physics-based Character Skills. *ACM Trans. Graph.* 37, 4 (July 2018), 143:1–143:14.

[28] David Pollard. 2000. Asymptopia: an exposition of statistical asymptotic theory. In *Asymptopia: an exposition of statistical asymp-totic theory*.

[29] Alex Ray, Joshua Achiam, and Dario Amodei. 2019. Benchmarking Safe Exploration in Deep Reinforcement Learning. In *OpenAI Blog*.

[30] Julien Roy, Roger Girgis, Joshua Romoff, Pierre-Luc Bacon, and Chris J Pal. 2022. Direct Behavior Specification via Constrained Reinforcement Learning. In *ICML*, Vol. 162. 18828–18843.

[31] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International conference on machine learning*. PMLR, 1889–1897.

[32] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[33] David Silver, Satinder Singh, Doina Precup, and Richard S. Sutton. 2021. Reward is enough. *Artificial Intelligence* 299 (2021), 103535.

[34] Adam Stooke, Joshua Achiam, and Pieter Abbeel. 2020. Responsive Safety in Reinforcement Learning by PID Lagrangian Methods. In *ICML*.

[35] L. Tai, G. Paolo, and M. Liu. 2017. Virtual-to-real DRL: Continuous control of mobile robots for mapless navigation. In *IROS*.

[36] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium (USENIX Security 18)*. 1599–1614.

[37] Xiaodong Yang, Tom Yamaguchi, Hoang-Dung Tran, Bardh Hoxha, Taylor T. Johnson, and Danil Prokhorov. 2022. Neural Network Repair with Reachability Analysis. In *Formal Modeling and Analysis of Timed Systems*. 221–236.

[38] Oleksii Zhelo, Jingwei Zhang, Lei Tai, Ming Liu, and Wolfram Burgard. 2018. Curiosity-driven exploration for mapless navigation with deep reinforcement learning. *arXiv preprint arXiv:1804.00456* (2018).