Verifying Online Safety Properties for Safe Deep Reinforcement Learning

Verifying Online Safety Properties for Safe Deep Reinforcement Learning

LUCA MARZARI, Department of Computer Science, University of Verona, Italy
FERDINANDO CICALESE, Department of Computer Science, University of Verona, Italy
ALESSANDRO FARINELLI, Department of Computer Science, University of Verona, Italy
CHRISTOPHER AMATO, Khoury College of Computer Sciences, Northeastern University, USA
ENRICO MARCHESINI, Laboratory for Information & Decision Systems, Massachusetts Institute of Technology, USA

Ensuring safety in reinforcement learning (RL) is critical for deploying agents in real-world applications. During training, current safe RL approaches often rely on indicator cost functions that provide sparse feedback, resulting in two key limitations: (i) poor sample efficiency due to the lack of safety information in neighboring states, and (ii) dependence on cost-value functions, leading to brittle convergence and suboptimal performance. After training, safety is guaranteed via formal verification methods for deep neural networks (FV), whose computational complexity hinders their application during training. We address the limitations of using cost functions via verification by proposing a safe RL method based on a *violation* value—the risk associated with policy decisions in a portion of the state space. Our approach verifies *safety properties* (i.e., state-action pairs) that may lead to unsafe behavior, and quantifies the size of the state space where properties are violated. This violation value is then used to penalize the agent during training to encourage safer policy behavior. Given the NP-hard nature of FV, we propose an efficient, sample-based approximation with probabilistic guarantees to compute the violation value. Extensive experiments on standard benchmarks and real-world robotic navigation tasks show that violation-augmented approaches significantly improve safety by reducing the number of unsafe states encountered while achieving superior performance compared to existing methods.

CCS Concepts: • **Computing methodologies** → **Reinforcement learning**; *Online learning settings*.

Additional Key Words and Phrases: Safe Deep Reinforcement Learning, Safe Intelligent Systems, Formal Verification for Deep Neural Networks

1 INTRODUCTION

In recent years, deep reinforcement learning (DRL) algorithms have demonstrated promising results in various applications, from video games [60] to robotic tasks such as manipulation [41, 48] and mobile navigation [31, 32, 57]. However, using these deep neural networks (DNNs)-based systems on real-world problems requires to incorporate safety into DRL methods.

Authors' Contact Information: Luca Marzari, luca.marzari@univr.it, Department of Computer Science, University of Verona, Verona, Italy; Ferdinando Cicalese, ferdinando.cicalese@univr.it, Department of Computer Science, University of Verona, Verona, Italy; Alessandro Farinelli, alessandro.farinelli@univr.it, Department of Computer Science, University of Verona, Verona, Italy; Christopher Amato, c.amato@northeastern.edu, Khoury College of Computer Sciences, Northeastern University, Boston, USA; Enrico Marchesini, emarche@mit.edu, Laboratory for Information & Decision Systems, Massachusetts Institute of Technology, Cambridge, USA.

ACM 2157-6912/2025/9-ART
https://doi.org/10.1145/3770068

Safe DRL approaches have been explored to enhance safety by modeling a problem as a constrained Markov decision process (CMPD) [4]. In a CMDP, an agent aims to maximize a reward signal while keeping an accumulated cost, received from visiting unsafe states, below a specified threshold. However, defining informative cost functions is as challenging as designing rewards [16] due to the difficulty of quantifying the risk of policy decisions around unsafe states. Consequently, the proposed solutions rely on indicator cost functions where a positive value marks a state-action pair (s, a) as unsafe [26]. These cost values are then backed up to propagate safety information and estimate a cost-value function, which guides the learning process toward safety via penalties [58] and constraints [25, 55]. However, learning cost-value functions with DNNs introduces brittle convergence properties and local optima [18, 28], limiting sample efficiency and the efficacy of cost functions in promoting safety. The sparse nature of the indicator cost also poses a significant issue, requiring numerous unsafe state visits to learn reasonable estimates from sparse feedback. Moreover, these sparse values also fail to provide information about the likelihood of incurring an unsafe behavior for a specific action a around s. Consider robotic navigation as an explanatory domain. In this task, the RL policy determines the agent's velocity based on sensor data, and prior works trigger a positive cost when an action results in a collision (Figure 1 on the left), deeming the state-action pair as unsafe. These methods thus necessitate many similar unsafe interactions around s to approximate the cost-value function effectively [10, 25, 55].

On top of these issues, DNNs are also vulnerable to small input variations that can fool a network to output an undesired value (or action) [56]. Although these inputs are most commonly observed in high-dimensional visual domains, Amir et al. [5] shows that they can also occur in low-dimensional state configurations like DRL navigation tasks, leading to unsafe behaviors and potential hazards. Formal verification (FV) for DNNs [23] has arisen to tackle this issue, leveraging state-action relationships (called *safety properties*) to provably detect these configurations. However, applying FV during training presents many challenges: (i) the *safety properties* are hand-designed by a system designer, which may be unfeasible in complex tasks; (ii) FV is NP-complete and thus computationally demanding [21]. These challenges raise the question: how can we enhance safety during training without relying on cost-value functions or the limitations of FV?

We propose a novel approach that leverages online safety properties (i.e., collected during training) to (i) get information on which part of the state space is potentially unsafe, and (ii) compute the probability of incurring in one of these configurations in the surrounding of that state. We then use this probability as a safety metric, called *violation*, as a penalty to the agent for discouraging unsafe decisions [1, 22]. We hypothesize that this procedure will significantly improve safety, reducing the number of hazardous states visited and addressing all the issues related to cost functions. Recalling the navigation example, let us consider a state s where performing an action a led to a collision. Our approach exploits state-action information (e.g., robot size, max velocity) to define an area of a specified size around s where performing a would likely result in unsafe interactions (Figure 1 on the right). Our goal is to quantify the states in the unsafe area where the policy chooses a and use this information as a penalty to bias the policy towards safer regions.

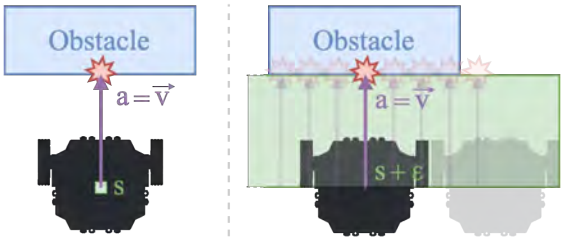


Fig. 1. Indicator cost function (left). Unsafe interactions are caused by the same action around the unsafe state (right).

To this end, in this paper, we combine and extend our recent works in bridging the gap between formal verification and safe DRL:

- Collection and refinement of online properties (CROP) framework [39] for collecting safety properties during training without relying on a system designer. CROP also refines *similar* properties to limit their number.
- Monte Carlo sample-based verification [33] that samples and propagates a set of states from the ϵ area of interest to approximate the violation value by counting the instances where the agent selects an unsafe action.

In detail, we significantly extend these contributions by providing probabilistic guarantees on the approximate violation value. We also analyze the non-Markovian nature of the violation penalty, presenting a state-augmented method to address such an issue. Additionally, we expand the empirical analysis by evaluating our framework on standard safe DRL benchmarks (i.e., SafeMuJoCo and SafetyGym [49]), and additional baselines. Finally, we transfer our trained policies to real robots in a navigation task [30, 31, 64].

In all scenarios, we apply the violation penalty to PPO [51], comparing to a cost-penalty version [51, 60], a constrained implementation based on the Lagrangian method (LPPO) [55], a Lagrangian-based penalty function (RCPO) [58], and the more recent penalty-based algorithm Penalized PPO (P3O) [65]. Our extensive evaluation shows that cost penalties result in high costs and violations at convergence, confirming the limitations of indicator cost functions. Lagrangian and penalty-based baselines exhibit similar issues due to using additional cost-value functions. In contrast, our violation-based penalty significantly reduces unsafe behaviors (i.e., lower cost and violation) while preserving high returns. To further assess the improvement in terms of safety provided by our proposed approach over different methods tested in this work, we compare the violation value for converged policies using a formal verification tool. Our results confirm the safety benefit, resulting in the lowest violation value percentage, confirming the impact of incorporating an approximate violation value as a penalty for the trained agents.

2 PRELIMINARIES

A DRL problem is typically modeled as a Markov decision process (MDP), described by a tuple $< \mathcal{S}, \mathcal{A}, T, r, \gamma >$ where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a state transition probability, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, and $\gamma \in [0, 1)$ is a discount factor. Given a policy $\pi \in \Pi := \{\pi(a|s) : s \in \mathcal{S}, a \in \mathcal{A}\}$, an RL agent aims to maximize the expected discounted return for each trajectory $\tau = (s_0, a_0, r_0, \dots)$ defined as follows:

$$\max_{\pi \in \Pi} J_r^\pi := \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]. \tag{1}$$

A common approach for safe DRL is to apply penalties to r, learning how to avoid unsafe interactions characterized by low returns [15]. Alternatively, the safety problem is formalized as a CMDP incorporating an additional finite set of constraints C, defined on $c_{1,\dots,n} : \mathcal{S} \times \mathcal{A} \to \{0, 1\}$ cost functions (where n is the number of constraints) and their thresholds $t_{1,\dots,n}$ [4]. Given a cost function c_i, the *cost return* is defined as $J_{c_i}^\pi := \mathbb{E}_{\tau \sim \pi}[\sum_{t=0}^{\infty} \gamma^t c_i(s_t, a_t)]$. Constraint-satisfying feasible policies Π_C, and optimal policies π^* are thus defined as:

$$\Pi_C := \{\pi \in \Pi : J_{c_i}^\pi \leq t_i, \ \forall i \in [1, \dots, n]\}, \quad \pi^* = \max_{\pi \in \Pi_C} J_r^\pi. \tag{2}$$

In practice, constrained DRL algorithms aim at maximizing the expected return J_r^π while maintaining costs under hard-coded thresholds $\mathbf{t} := t_{1,\dots,n}$:

$$\max_{\pi \in \Pi} J_r^\pi \quad \text{s.t.} \quad J_C^\pi \leq \mathbf{t}. \tag{3}$$

For computational efficiency, such a constrained optimization problem is usually transformed into an equivalent unconstrained one using the Lagrangian method [47, 50, 55]. However, RCPO [58] highlighted significant limitations in these constrained setups, such as the need for policy parametrization (incompatible with value-based DRL) and the necessity to propagate constraint violations over trajectories. RCPO effectively handles different types of constraints (e.g., cumulative, instantaneous) by integrating a Lagrangian-based penalty signal into the reward function. Similarly, P3O uses a simple penalty based on the gap between an additional cost value function and desired safety thresholds, to eliminate cost constraints. For these reasons, RCPO and P3O are the works most closely related to ours. These insights also motivate our use of penalty-based approaches to evaluate the benefits of incorporating the violation value as a penalty to foster safety. Nonetheless, to provide a comprehensive evaluation, our experiments will include both the Lagrangian implementation of the PPO algorithm, RCPO, and P3O as baselines for comparison.

2.1 Safety Properties and Formal Verification

A safety property for a DNN ($\mathcal{N}$) is typically expressed as a state-action relationship [23, 61]. Specifically, a property is encoded using a precondition and a postcondition. The precondition is a predicate $\mathcal{I}$ on the state features modeling desired unsafe situations (i.e., the domain of the property). It is used to check if a specific state falls within a set of intervals $\mathcal{X}$ representing the unsafe area of interest. Thus, $\mathcal{X}$ is a Cartesian product of intervals, one for each feature in the state space. The postcondition is a predicate Q on the output of $\mathcal{N}$ specifying the action to avoid in $\mathcal{X}$. Broadly speaking, the postcondition Q encodes a requirement such as *never select the action corresponding to the output* y_i, with i being the index of the action to avoid.

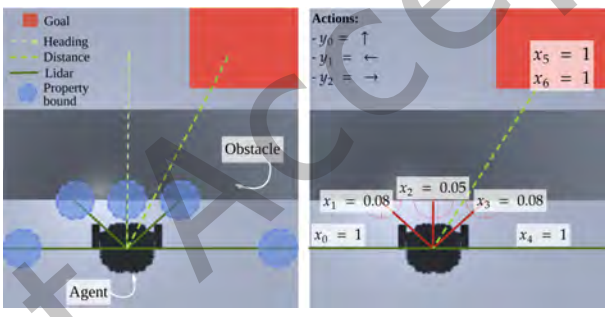


Fig. 2. Left: components of a safety property (legend in the upper left corner). Right: Explanatory image of a safety property for a navigation context.

Using the navigation task, Figure 2 shows an explanatory encoding of a safety property in a value-based DRL setup, where the policy selects the action with the highest value.[1] To encode a property, we start by considering the current state of the agent as a vector of observed features $x_0, \ldots, x_6$, and its possible actions $\{y_0, y_1, y_2\}$ corresponding to a forward, left, and right movement. Given there is an obstacle in front of the robot, the safety property uses an $\epsilon = 0.05$ surrounding of the agent's state (the blue bounds on the left figure) to model the set of intervals $\mathcal{X}$ representing the unsafe area (i.e., where a forward action would lead to a collision). In this situation,

[1] Similar considerations apply to the continuous policy-gradient case by using deterministic policies and checking the action values, and not their max.

we want to check that a forward movement is not selected, so a safety property is formally encoded as follows:

$$\underbrace{\forall\, x_0, x_4 \in [0.95, 1], x_1, x_3 \in [0.03, 0.08], x_2 \in (0, 0.05], x_5 \in (0, 1], x_6 \in [-1, 1],}_{\mathcal{I}}\ \underbrace{\max(y_1, y_2) > y_0,}_{\mathcal{Q}}$$

where $x_0, \ldots, x_4$ are the 5 lidar values, x_5, x_6 is the relative position of the agent with respect to the goal (i.e., distance and heading), and action y_0 corresponds to the forward movement. In the example, we set $\mathcal{X}_5 = (0, 1], \mathcal{X}_6 = [-1, 1]$ to indicate a generic goal position in the environment. Hence, if the agent is in any state that satisfies $\mathcal{I}$, i.e., $\forall\, \mathbf{x} = [x_0, x_1, x_2, x_3, x_4, x_5, x_6]^{\mathrm{T}} \in \mathcal{X}$ (with $\mathcal{X} = \{[0.95, 1], [0.03, 0.08], (0, 0.05], [0.03, 0.08], [0.95, 1], (0, 1], [-1, 1]\}$), if $x_0 \in [0.95, 1], x_1 \in [0.03, 0.08], x_2 \in (0, 0.05], x_3 \in [0.03, 0.08], x_4 \in [0.95, 1], x_5 \in (0, 1], x_6 \in [-1, 1]$, it should not select the forward action y_0.

2.1.1 DNN-Verification and the Violation Value. An instance of the FV problem for decision-making is called *DNN-Verification*. The problem is represented by a tuple $\mathcal{R}$ composed by $\mathcal{N}$, and a safety property $\langle \mathcal{I}, \mathcal{Q} \rangle$ and is formally defined as follows:

Definition 2.1 (DNN-Verification problem).
Input: A tuple $\mathcal{R} = \langle \mathcal{N}, \mathcal{I}, \mathcal{Q} \rangle$, where $\mathcal{N}$ is a DNN, $\mathcal{I}$ is a precondition on the input, and $\mathcal{Q}$ a postcondition on the output.
Output: SAT if $\exists\, x \mid \mathcal{I}(x) \wedge \mathcal{Q}(\mathcal{N}(x))$ and UNSAT otherwise, indicating that no such x exists.

DNN-Verification aims to answer *whether in the domain of the property, it exists at least one input configuration x for $\mathcal{N}$ (i.e., a state) that satisfies both the predicates $\mathcal{I}$ and $\mathcal{Q}$*. State-of-the-art FV methods either use interval analysis [45] to propagate input bounds represented by $\mathcal{X}$ through the network and perform reachability analysis in the output layer, or encode linear combinations and the non-linear activation functions of a DNN as constraints for an optimization problem [23, 40].

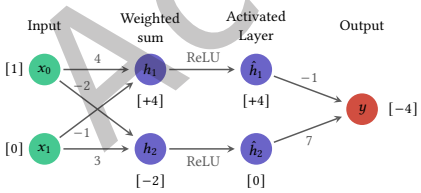


Fig. 3. Explanatory example of DNN-Verification.

Figure 3 shows an explanatory example of how to check the existence of an unsafe input configuration. Suppose we aim to verify whether the depicted toy-network $\mathcal{N}$ outputs a value ≥ 0 for any input in the interval $[0, 1]$. We define $\mathcal{I}$ as the predicate on the input vector $\mathbf{x} = [x_0, x_1]^T$ which is true iff $\mathbf{x} \in [0, 1] \times [0, 1]$, i.e., $\mathcal{X} = \{[0, 1], [0, 1]\}$ and $\mathcal{Q}$ as the predicate on the output y which is true iff $y = \mathcal{N}(x) < 0$, that is, we set $\mathcal{Q}$ to be the negation of the property. Then, solving the verification problem on the instance $(\mathcal{N}, \mathcal{I}, \mathcal{Q})$ returns SAT iff there is a counterexample that violates our property. Since the input vector $\mathbf{x} = [1, 0]^T$ leads to an output < 0, the result of a FV tool is SAT—there is at least one configuration x that satisfies $\mathcal{I}$ and for which $\mathcal{N}(x) < 0$. As a result, we can say that the network is unsafe with respect to that property.

2.1.2 #DNN-Verification. The binary answer returned by DNN-Verification, however, does not quantify the number of violations in the area of interest $\mathcal{X}$—the probability of selecting an unsafe action in $\mathcal{X}$. Marzari et al. [35] recently address this issue by formalizing the *#DNN-verification* problem as follows:

Definition 2.2 (#DNN-Verification problem).
Input: A tuple $\mathcal{R} = \langle \mathcal{N}, \mathcal{I}, \mathcal{Q} \rangle$, as in Definition 2.1.
Output: $|\Gamma(\mathcal{R})|$

with $\Gamma(\mathcal{R}) = \left\{ x \mid \mathcal{I}(x) \wedge \mathcal{Q}(\mathcal{N}(x)) \right\}$ representing the set of all the input configurations for $\mathcal{N}$ satisfying the property defined by $\mathcal{I}$ and $\mathcal{Q}$ (i.e., all the unsafe state-action pairs). Since our setting is in the continuum, the number of points in any non-empty set is infinite. Hence, we consider the cardinality as a proxy for the volume of the corresponding set. Nonetheless, if we discretize the space to the machine precision, $\Gamma(\mathcal{T})$ becomes a finite countable set. In practice, rather than the cardinality of $\Gamma(\mathcal{R})$, it is more useful to define the problem in terms of the ratio between the cardinality of Γ and the one of the set of inputs satisfying $\mathcal{I}$. This quantity is called *violation* and it is formally defined as follows:

Definition 2.3 (Violation value). Given an instance of *DNN-Verification* $\mathcal{R} = \langle \mathcal{N}, \mathcal{I}, \mathcal{Q} \rangle$,

$$violation = \frac{|\Gamma(\mathcal{R})|}{|\{x \mid \mathcal{I}(x)\}|}.$$

We refer the interested reader to previous works for an exhaustive overview of FV and its extensions that tackle these problems [34, 35, 37]. Here, we remark that DNN-Verification is an NP-complete problem [21]. Moreover, formally counting (or enumerating) all violation points in an area of interest is even more challenging as it is a #P-hard problem [35]. Hence, these methods are unfeasible to be applied during training and motivate the investigation of approximate solutions.

3 SAFE DEEP REINFORCEMENT LEARNING VIA VERIFICATION

In this section, we present our two main contributions addressing the limitations of FV methods discussed above, namely: (i) the complexity of hand-design safety properties and (ii) the computational overhead of computing the violation value. To tackle the first issue, we propose the collection and refinement of online properties framework. To address the latter, we introduce a Monte Carlo sampling approach that approximates the violation value with probabilistic guarantees, providing an efficient alternative to the "formal" violation value computed using modern FV tools.

3.1 Collection and Refinement of Online Properties

Figure 4 shows an overview of the proposed CROP framework (in blue), which is designed to collect and refine safety properties during training. Similar to existing safe DRL methods [25, 50, 55], CROP uses a cost indicator function to identify unsafe state-action interactions. When such an interaction is detected, we generate a safety property using the state and the action that led to the hazardous situation. CROP then performs a refinement procedure merging properties deemed similar according to a similarity rule, which is described in the following section. Finally, the violation value is computed using the sample-based approximation described in the next section, applied to the properties generated by CROP.

The general flow of CROP is presented in Algorithm 1. We augment the training of a DRL algorithm with a buffer $\mathbf{P} = \{properties\ \mathcal{P}\}$ that stores the properties generated by CROP (line 3). Given a state s_t deemed unsafe by the cost signal, CROP generates a new safety property $\mathcal{P}'$ considering an initial $\boldsymbol{\epsilon}$ surrounding (passed as a parameter) of the state s_{t-1} and the action a_{t-1} that triggered the positive cost (line 5). In particular, we model the precondition $\mathcal{I}$ of the safety property with a set $\mathcal{X}$ of intervals encoded as $[x_i - \boldsymbol{\epsilon},\ x_i] \quad \forall\ x_i \in \mathbf{x}$, where x_i is the i-th input feature of the state s at timestep $t-1$. Notably, we encode the interval $[x_i - \boldsymbol{\epsilon},\ x_i]$ and not $[x_i - \boldsymbol{\epsilon},\ x_i + \boldsymbol{\epsilon}]$ to exclude potentially unfeasible configurations The postcondition $\mathcal{Q}$ checks that the agent does not select the action a_{t-1} that led to the unsafe state s_t. CROP then samples the properties whose set

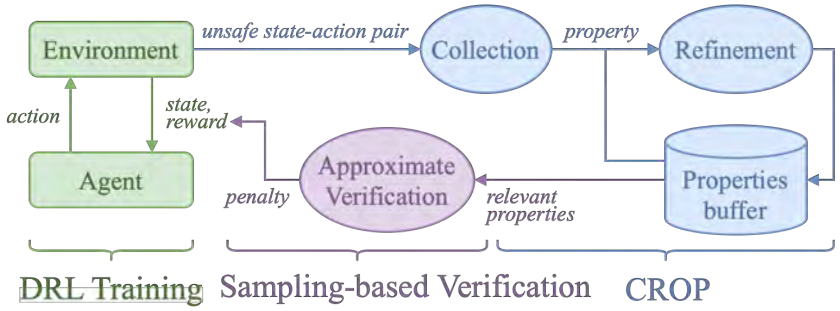


Fig. 4. Overview of the CROP framework.

$\mathcal{X}$ contains the state s_{t-1}, and whose postcondition Q considers the same unsafe action a_{t-1} that triggered the positive cost (line 6). For each selected property, we check if the new property $\mathcal{P}'$ is similar to a sampled one $\mathcal{P}$ that will be refined as described in the following section (lines 7-14). Notably, the $\boldsymbol{\epsilon}$ employed by CROP differs from the ϵ value typically required to design a safety property (as discussed in Section 3.3.1), which requires some knowledge of the task or agent's hardware. CROP's $\boldsymbol{\epsilon}$ is initialized as an arbitrarily small value that is reshaped during refinement. Moreover, we note in domains where many unsafe interactions occur, the buffer $\mathbf{P}$ grows unbounded. Hence, we reset $\mathbf{P}$ at the beginning of each episode (line 3). This approach allows us to compute the violation value only on unsafe situations encountered over the last epoch.

3.1.1 Similarity Rule and Refinement. Different safety properties could model a similar unsafe interaction (or not). We define a similarity rule to identify when two safety properties are *not similar*. To clarify the process, let us recall our navigation example. Consider two sets $\mathcal{X} = \{[0.95, 1], [0.95, 1], [0.01, 0.06], [0.03, 0.08], [0.95, 1]\}$, $\mathcal{X}' = \{[0.04, 0.09], (0, 0.05], [0.95, 1], [0.95, 1],$

Algorithm 1 Collection and Refinement of Online Properties

1: **Given:**
- $\mathcal{N} \leftarrow$ a DRL agent parametrized by a DNN
- $\boldsymbol{\epsilon} \leftarrow$ initial size for the area surrounding an unsafe state
- $\beta \leftarrow$ similarity threshold to merge properties.
- $\psi \leftarrow$ minimum interval that encodes a potentially unsafe situation.

2: During each episode of the training of DRL agent:
3: $\mathbf{P} \leftarrow \emptyset$
4: **if** s_t is unsafe (i.e., cost > 0) **then**
5: $\quad \mathcal{P}' \leftarrow$ `GenerateProperty`$(\boldsymbol{\epsilon},\ s_{t-1}, a_{t-1})$
6: $\quad$ **for** $\mathcal{P} \in \mathbf{P}$ where $s_{t-1} \subseteq \mathcal{X} \wedge a_{t-1}$ is unsafe in Q of $\mathcal{P}$ **do**
7: $\quad\quad$ **if** $\nexists$ `Similar`$(\mathcal{P}, \mathcal{P}', \beta, \psi)\ \forall \mathcal{P} \in \mathbf{P}$ **then**
8: $\quad\quad\quad \mathbf{P} \leftarrow \mathbf{P} \cup \mathcal{P}'$
9: $\quad\quad$ **else**
10: $\quad\quad\quad \mathbf{P} \leftarrow$ `PropertyRefinement`$(\mathcal{P}, \mathcal{P}', \mathbf{P})$
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: $\quad$ *violation* $\leftarrow$ `ApproximateVerification`$(\mathcal{N}, \mathbf{P})$ ▷ as in Section 3.2
14: **end if**

Fig. 5. Explanatory example of two different safety properties for the same forward action.

$[0.95, 1]\}$ encoding the preconditions of two safety properties $\mathcal{P}$ and $\mathcal{P}'$, respectively.[2] These scenarios are shown in Figure 5, where the agent would collide upon moving forward (i.e., the two safety properties have the same postcondition). The two preconditions significantly differ. Let us compare the first two intervals of $\mathcal{X}$ and $\mathcal{X}'$—$\mathcal{X}_0 = [0.95, 1]$ and $\mathcal{X}'_0 = [0.04, 0.09]$—as a practical example. We formally define the two properties to be *not similar* by leveraging Moore's interval algebra [45] and defining the following similarity rule:

$$\mathcal{P} \nsim \mathcal{P}' \iff \exists\, i \in \{0, |\mathcal{X}|\}\ s.t.\ \left(\mathcal{X}_i \subseteq \psi \vee \mathcal{X}'_i \subseteq \psi\right) \wedge |\mathcal{X}_i - \mathcal{X}'_i| > \beta.$$

Specifically, given the two safety properties $\mathcal{P}$ and $\mathcal{P}'$, we check: (i) whether there is at least one interval in the sets $\mathcal{X}$, $\mathcal{X}'$ contained in ψ. ψ is the smallest interval encoding a potentially unsafe situation (e.g., in Figure 5, $\psi = [0, 0.09]$ and is highlighted in red); (ii) whether the lower bound of the absolute value difference between the two intervals $|\mathcal{X}_i - \mathcal{X}'_i|$ is greater than a similarity threshold value β.[3] If the two conditions hold for at least one interval, then $\mathcal{P}$ and $\mathcal{P}'$ are deemed not similar. On the other hand, if two properties are *similar*, we perform a refinement process merging them by considering the minimum lower and maximum upper bound for each interval of $\mathcal{X}$ and $\mathcal{X}'$. The refinement enables us to cluster together similar unsafe situations. Figure 6 shows an explanatory refinement process in a scenario considering the forward action y_0.

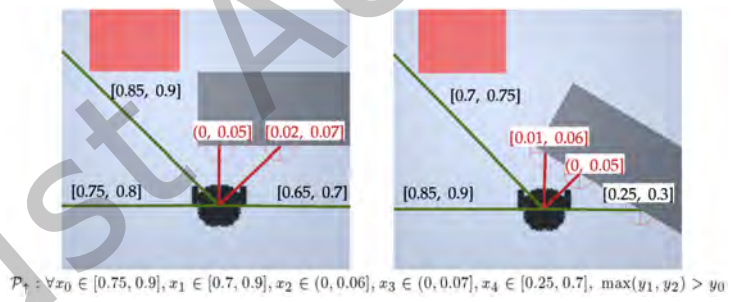


Fig. 6. Explanatory example of refinement process for two *similar* safety properties.

3.2 Monte Carlo Sampling-based Approximate Verification

The violation value computed on a safety property quantifies the probability of incurring unsafe policy decisions within an area of the state space, addressing the sparsity of indicator cost functions (Figure 4 in purple). However,

[2]For simplicity, we omit the values for the goal's heading and distance.
[3]β is a hyperparameter that can be tuned to be more or less restrictive in considering two properties as similar.

due to the NP-completeness and #P-hardness of verification [21, 35], computing a formal violation (i.e., using a FV tool) during training is intractable.

We propose a computationally efficient sample-based method to approximate the violation value, allowing us to use it as a training penalty. Similarly to the violation computed with formal tools, our approximation can be interpreted as a locally aware (dense) cost value. It quantifies the risk of performing a certain action a in a state s, based on the hazardous effects that a could have in a local region around s. In detail, consider a deep neural network $\mathcal{N}$ with $\{y_1, \ldots, y_n\}$ outputs representing the possible actions the agent can perform in the environment. Let y_k be the unsafe action to avoid in a particular region of interest $\mathcal{X}$ specified by the precondition $\mathcal{I}$. Our goal is to count the number of times $y_k > \max(y_i)\ \forall i \in \{1, \ldots, n\} \setminus \{k\}$ over the total number of states tested. This approximation offers many advantages over indicator cost functions: it (i) provides safety information for areas of interest as it is computed over state-action mappings; (ii) approximates how often a property violation might occur, having a similar role to Lagrangian multipliers without requiring additional gradient steps or value estimators; (iii) does not require additional environment interactions, significantly reducing the number of visited unsafe states.

Algorithm 2 shows the general flow of our sample-based method. Our approach requires $\mathcal{N}$, an unsafe state s, a property buffer $\mathbf{P}$, and the number of samples to use for computing the approximate violation value. The first step is to consider the safety properties that contain s in their preconditions, and store these properties in a new buffer $\mathbf{P'}$ (line 1). We initialize the violation value to zero, and for each $\mathcal{P} \in \mathbf{P'}$ we randomly sample a set of m input vectors for the DNN $\mathbf{X} = \{\mathbf{x_1}, \ldots, \mathbf{x_m}\}$ from $\mathcal{X}$ (lines 2-4). After propagating $\mathbf{X}$ through $\mathcal{N}$, we count how many outputs do not satisfy the postconditions (line 5). To this end, for each $\mathbf{x_i}$, we check if the index of the corresponding output, (i.e., the one with the highest value) is equal to the index k of the unsafe action to avoid. Finally, we compute our approximate violation value as the ratio between the number of such outputs over the total sampled states (line 7). This process closely resembles how the formal violation is computed in [35] but employs random sampling and feed-forward steps through the $\mathcal{N}$, avoiding the computational demands associated with FV tools.

Algorithm 2 Computing the Approximate Violation Value

Given $\mathcal{N}$ with outputs $y_1, \ldots, y_n$, unsafe state s, properties buffer $\mathbf{P}$, and m number of states to sample.

1: $\mathbf{P'} \leftarrow \langle I, Q \rangle$ if $s \cap \mathcal{X} \neq \emptyset,\ \forall\ \langle I, Q \rangle \in \mathbf{P}$ ▷ $\mathcal{X}$ encodes the precondition $\mathcal{I}$ of the property $\mathcal{P} = \langle I, Q \rangle$ under consideration
2: *violation* $\leftarrow 0$
3: **for** $\mathcal{P} \in \mathbf{P'}$ **do**
4: $\quad \mathbf{X} \leftarrow \texttt{Sample}(m, \mathcal{X})$
5: $\quad$ *violation* $\leftarrow$ *violation* + $\texttt{count}(\arg\max(\mathcal{N}(\mathbf{X})) = k)$ ▷ considering y_k deemed unsafe in the postcondition Q of $\mathcal{P}$
6: **end for**
7: **return** *violation* / $(m\ |\mathbf{P'}|)$

3.2.1 Visual Example. We further clarify the approximate violation value computation by relying on our navigation example in Figure 7. The agent receives 5 lidar scan values as input and outputs three values $\{y_1, y_2, y_3\}$ corresponding to a forward, left, and right movement, respectively. We thus have $\mathcal{X} = \{[0.75, 0.8], [0.85, 0.9], (0, 0.05], [0.02, 0.07], [0.65, 0.7]\}$ encoding the unsafe area where we want to avoid the action y_1 that has index 0 (i.e., a forward movement at velocity $\vec{v}$). Hence, given any state sampled in $\mathcal{X}$, the action y_1 should not be selected. This safety property is formally defined as $\forall \mathbf{x} \in \mathcal{X}$, $\max(y_2, y_3) > y_1$.

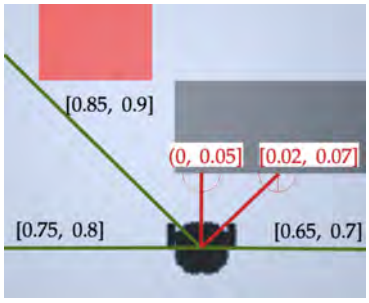


Fig. 7. Visual example of a potential unsafe situation for the DRL agent.

Considering a sample of $m = 2$ states from the $\mathcal{X}$ of the property precondition, we obtain two states $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2\} \in \mathcal{X}$ which are then forward propagated through $\mathcal{N}$. If this propagation outputs $\mathcal{N}(\mathbf{x}_1) = [1, 3, 2]$, then the greedy agent selects $\arg\max(\mathcal{N}(\mathbf{x}_1)) = 1$. Conversely, for $\mathbf{x}_2$ we obtain $\mathcal{N}(\mathbf{x}_2) = [4, 2, 3]$, and the agent selects $\arg\max(\mathcal{N}(\mathbf{x}_2)) = 0$. To approximate the violation value, we count the states $\mathbf{x} \in \mathcal{X}$ where the argmax of $\mathcal{N}(\mathbf{x})$ is equal to the index of the unsafe action y_k, in this case 0. In this example, we have one violation since only the second propagation of $\mathbf{x}_2$ leads to $\arg\max[\mathcal{N}(\mathbf{x}_2)] = 0$. By normalizing the number of violations over the size of the sampled configurations, we obtain the violation value of 0.5, meaning that half of all the states sampled led to a property violation. A natural question that arises is what number of states $\mathbf{x}_m$ is needed to get an accurate estimate of the violation and what kind of guarantees we obtain about this estimate.

3.3 Probabilistic Guarantees on the Violation Value

We aim to estimate the number of states where a binary property holds (i.e., whether the agent commits a violation or not) from a potentially infinite set of states. To this end, we can sample a smaller subset of the states and show that this particular subset is representative of the larger one we are interested in. To this end, we use the following Chernoff bound.

Theorem 3.1 (Chernoff bound). *Let $X_1, \cdots, X_m$ be independent indicator random variables, such that $Pr[X_i = 1] = p_i$. Let $X = \sum_{i=1}^{m} X_i$ and $\mu = \mathbb{E}[X] = \sum_{i=1}^{m} p_i$. For $0 < \epsilon < 1$ the following inequality holds:*

$$Pr[|X - \mu| \geq \epsilon\mu] \leq 2e^{-\frac{\mu\epsilon^2}{3}}.$$

A direct application of Theorem 3.1 implies the following result (see, e.g., [44, section 4.2.1]), that we recall here to keep the paper self-contained.

Let p be the actual violation value in the area $\mathcal{X}$ that encodes the property's precondition. Assume we would like to estimate p up to an additive error-tolerance θ and with confidence at least $1 - \delta$, for fixed (desired) $\delta, \theta \in (0, 1)$. To this end, we sample m points $\mathbf{x_1}, \ldots, \mathbf{x_m}$ independently and uniformly from the property input space $\mathcal{X}$. For each $i = 1, \ldots, m$, let X_i be the indicator random variable of the event that $\mathbf{x}_i$ is a violation point, hence $Pr[X_i = 1] = p$. Define $\overline{X} = \frac{\sum_{i=1}^{m} X_i}{m}$. We are interested in a bound on m that can guarantee $Pr[|\overline{X} - p| \leq \theta] \geq 1 - \delta$.

Let $X = \sum_{i=1}^{m} X_i$, hence $\mu = \mathbb{E}[X] = mp$. Considering that $p \leq 1$, by Theorem 3.1 we get

$$Pr[|\overline{X} - p| \geq \theta] = Pr[|X - \mu| \geq m\theta] = Pr[|X - \mu| \geq \mu\frac{\theta}{p}] \leq 2e^{-\frac{\mu\theta^2}{3p^2}} = 2e^{-\frac{m\theta^2}{3p}} \leq 2e^{-\frac{m\theta^2}{3}}.$$

Therefore, for

$$m = \frac{3}{\theta^2} \ln\left(\frac{2}{\delta}\right), \tag{4}$$

we have

$$Pr[|\overline{X} - p| \geq \theta] \leq \delta.$$

Recalling that we are interested in the complementary event of the term on the left-hand side of the latter inequality, the above derivation leads to the following result.

Proposition 3.2. *Fix $\theta, \delta \in (0,1)$, and let m be an integer satisfying $m \geq \frac{3}{\theta^2} \ln(\frac{2}{\delta})$. Let $X_1, \cdots, X_m$ be independent and identically distributed indicator random variables for the event that a uniformly sampled input point from the set $\mathcal{X}$ violates the property, and let $X = \sum_{i=1}^{m} X_i$, whence $Pr[X_i = 1] = p$, and $\mu = \mathbb{E}[X] = mp$. Then, $Pr[|\frac{X}{m} - p| \leq \theta] \geq 1 - \delta$.*

The proposition gives a lower bound on the number of m points in $S = \mathbf{x}_1, \ldots, \mathbf{x}_m$ such that if V are violation points, $|V|/m$ provides an estimate of p with the desired precision θ and confidence δ. For example, if we want a maximum error tolerance of $\theta = 0.05$ with confidence at least $1 - \delta = 0.99$ for the approximate violation we are computing, sampling $m \geq \frac{3}{\theta^2} \ln(\frac{2}{\delta}) = \frac{3}{0.05^2} \ln(\frac{2}{0.01}) = 6357$ states would be sufficient.

It is important to note our approach is not efficient when the formal violation value we are trying to estimate is arbitrarily small. To clarify this, let M be the size of the instance (the network's encoding and the property where we want to compute the violation value). If $p \geq \frac{1}{poly(M)}$ (i.e., bounded by some polynomial function of the instance size), the approach is clearly a polynomial time algorithm in M as it requires a number of samples polynomial in the size of the instance. On the other hand, if, for instance, $p = o(2^{-M})$, then our sampling method would require an exponential number of samples. However, in view of the hardness of both the FV problem and its counting version, under the standard complexity assumptions, we believe this intractability issue is strictly related to the inherent complexity of the problem rather than a weakness of our approach. From (4), we see that for any fixed m, there is a direct proportionality between the error tolerance θ and the values of the confidence $c = 1 - \delta$, that is, $\theta = \sqrt{\frac{3}{m} \ln\left(\frac{2}{1-c}\right)}$. Hence, for any fixed m, the better (higher) the confidence $c = 1 - \delta$, the looser (higher) the error tolerance bound θ. Figure 8 visualizes this relationship. From the curves, we note that using $m = 10000$ sample is sufficient to guarantee confidence ≥ 0.99 and an error tolerance below 0.05.

3.3.1 Comparing the Approximate Violation with the Formal One. To empirically assess how good our approximate violation is, we compare it over the exact violation computed by ProVe, a FV tool for the #DNN-Verification [35].[4]

To this end, we consider the same navigation task used in our previous examples, the average results of ten models collected at random steps during the training, and the following set of hand-designed properties:[5]

- $\mathcal{P}_{\uparrow}$: *There is an obstacle close in front ⇒ Do not go forward*
- $\mathcal{P}_{\leftarrow}$: *There is an obstacle close to the left ⇒ Do not turn left*
- $\mathcal{P}_{\rightarrow}$: *There is an obstacle close to the right ⇒ Do not turn right*

Table 1 shows a comparison between the violation value for $\mathcal{P}_{\uparrow,\leftarrow,\rightarrow}$ computed by ProVe, and with our approximation with samples of size 100, 1000, and 10000. We measure the difference between the formal violation value (i.e., first column) and our best approximation achieved with $m = 10000$ states (i.e., last column). These values are computed using the three safety properties, and result in a negligible 0.47% difference. In practice, our sample-based method achieves an accurate estimation of the real average violation value. Moreover, by exploiting

[4]ProVe is available at https://github.com/Isla-lab/Neural-Network-Verifier

[5]We use the maximum agent velocity to determine the size of the area around the (unsafe) states of interest (i.e., the $\boldsymbol{\epsilon}$-area). However, for simplicity, we describe these properties in natural language.

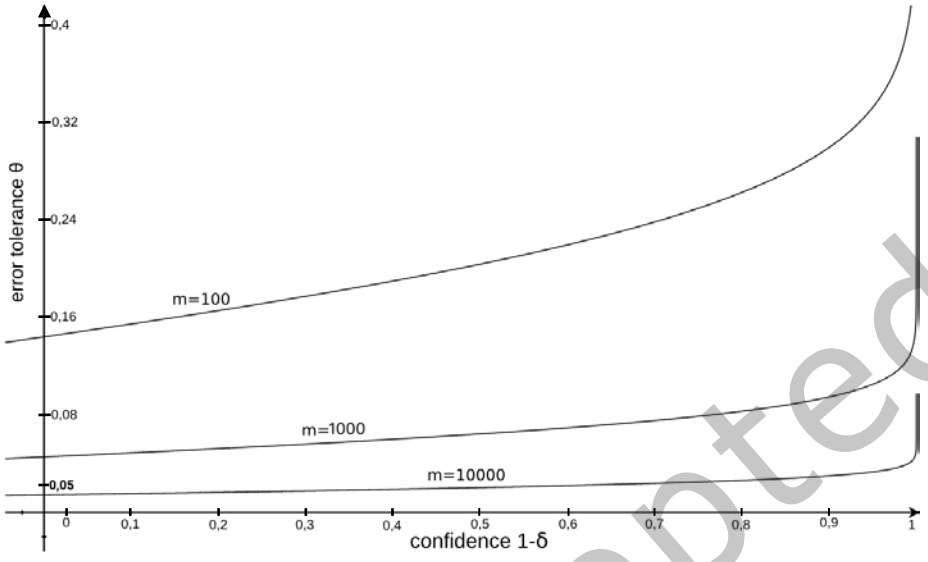


Fig. 8. The relation between confidence (x-axis) and error tolerance (y-axis) for different values of $m = 100, 1000, 10000$.

parallelism and batch computation of modern deep learning frameworks, the increase in computation time for the approximate violation with 100 or 10000 samples is comparable. Conversely, ProVe's average computation time is orders of magnitude higher compared to our approach (i.e., 0.06 over 157 seconds).

Table 1. Average *violation* (%), and computation time for properties $\mathcal{P}_{\uparrow,\leftarrow,\rightarrow}$ calculated using ProVe, and our approximation with 100, 1000, and 10000 samples.

Property	**ProVe**	**Estimation 100**	**Estimation 1k**	**Estimation 10k**
$\mathcal{P}_{\uparrow}$	81.48 ± 1.2	81.0 ± 0.6	81.1 ± 0.8	81.17 ± 0.5
$\mathcal{P}_{\leftarrow}$	73.9 ± 0.8	73.5 ± 0.2	73.63 ± 0.2	73.65 ± 0.3
$\mathcal{P}_{\rightarrow}$	74.2 ± 0.3	73.6 ± 0.1	73.67 ± 0.1	73.68 ± 0.1
Mean violation:	**76.53**	76.00	76.13	**76.17**
Mean computation time:	**≈2m37s**	**≈0.053s**	**≈ 0.056s**	**≈0.060s**

4 EMPLOYING THE VIOLATION VALUE TO FOSTER SAFETY

Having an efficient way for collecting properties and computing a good approximation of the violation, we now investigate the benefits of using the violation as a penalty during training.

Following prior penalty-based approaches [15, 58], we maximize the following objective:

$$\max_{\pi \in \Pi} J_r^\pi := \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) - Z(\cdot) \right] \tag{5}$$

where $Z(\cdot)$ is a penalty function. For instance, $Z(\cdot)$ could be expressed as $Z(s, a) = -1$ upon experiencing a collision, i.e., as a cost function. In our case, we employ a violation-penalty $Z_\pi(s_t \pm \boldsymbol{\epsilon}) = \mathbb{E}_{s \sim s_t \pm \boldsymbol{\epsilon}} \mathbb{E}_{a_t \sim \pi(s)} \mathbb{1}[a_t = a_k]$, where a_k is the unsafe action that led to a collision. Such a value indicates that the violation depends on the policy decisions in a proximity $\boldsymbol{\epsilon}$ of the state (i.e., the $\boldsymbol{\epsilon}$-area, or R). Equation 5 has two main benefits over other constrained DRL algorithms:

- Penalty-based objectives potentially maintain the same optimal policy of the underlying MDP as they do not constrain exploration nor reduce the space of feasible policies as constrained approaches [46].[6]
- Constrained DRL typically estimates an additional value function to propagate cost information, hindering their application with values that strictly depend on the current policy. This also requires visiting unsafe states to learn effective estimates for the sparse cost values. In contrast, penalty-based methods do not require learning additional value functions. Additionally, many DRL algorithms, such as PPO and RCPO, provide significant empirical evidence on the benefits of employing penalties over constraints [58].

5 LIMITATIONS

This work only considers vectorized inputs for computing the violation. While it is conceptually feasible to include different modalities (e.g., images, temporal sequences), doing so would require further research and experiments. In more detail, while related verification works [43, 52] have been previously employed in vision-based problems [8], they typically consider robustness analysis rather than verifying image-based control policies. This stems from the fact that it is not trivial to apply a perturbation that would change the state of the agent (i.e., what it sees) coherently without additional strong assumptions, such as having access to vision models of the world. For example, applying a perturbation to an Atari game taken as input by a policy would result in a noisy image, while verifying the policy behaviors requires encoding images of related state configurations. Hence, despite being conceptually feasible considering different modalities for control policies, applying the proposed approach to these settings is seldom straightforward and goes beyond the scope of our paper. Another limitation of our approach is related to the fact that, while our framework addresses the issues of using indicator cost functions and constraints, it remains unclear how to provide provable safety guarantees in model-free, safe DRL training approaches. Such a lack of formal guarantees on the behavior of the policy during training includes our work, constrained DRL [2, 25, 62], and several other approaches summarized in [15]. As discussed in [18], using DNNs to approximate policies and values causes the method to diverge from the underlying theoretical framework. Nonetheless, we begin to address these key issues by employing approximate violation with probabilistic guarantees during training, and formal verification approaches at convergence to check the trained policy decisions over the properties of interest.

6 EMPIRICAL EVALUATION

In this section, we propose an empirical evaluation to assess the benefits of employing our violation penalty for training DRL agents. To this end, we used a realistic set of navigation tasks for a Turtlebot3 mobile robot as well as standard safety benchmarks inspired to [49]. Our experiments aim to show the following: (i) The benefits of combining the reward signal with our provable safety metric to enhance the learning of safer behaviors at convergence. (ii) The advantages of using the proposed violation metric over indicator cost functions. (iii) The

[6] This is not the case for navigation tasks as safe behaviors bias the policy to avoid obstacles.

effectiveness of our additional investigations: using CROP over commonly employed hand-designed properties; relying on a state-augmentation method to deal with the issues related to the Markov property.
To assess our claims, the following plots consider a variety of metrics such as *success* as the number of goals reached, *cost* as the number of the agent collides with an obstacle, and finally, the violation at different stages of the training. All such metrics are averaged over the last 1000 steps. Each methodology uses CROP to collect online properties during training to compare the violation metric. The average violation is thus calculated on the properties collected by each methodology (but clearly, we used the violation value as a penalty only in our approach).

The data are collected on an i7-9700k and consider the mean and standard deviation of ten independent runs [12]. We consider the cost and violation penalty objective (Equation 5) in a policy-based (PPO [51]) baselines, referring to the resultant algorithms as PPO_{cost, violation}. Hence, PPO_cost receives a fixed penalty $Z = 1$ upon experiencing an unsafe interaction (i.e., the cost). This penalty value is based on the results obtained in the additional experiment of [33], where we investigate the effects of different penalty regularization terms, Z. In particular, we found a good trade-off between performance and safety with $Z = 1$, while lower values give more importance to the reward, resulting in more unsafe behaviors. We compare with Lagrangian PPO, LPPO [55], as it is a widely adopted constrained DRL baseline and achieves state-of-the-art performance in similar SafetyGym navigation-based tasks[7] In addition, we compare our approach with RCPO, which employs a reward penalty based on Lagrangian multipliers [58] and P3O that relies on a scaled penalty based on the gap between the estimated cost and a desired safety threshold. As such, these works are the most closely related methods to ours.

According to relevant literature, the value-based and policy-gradient baselines should achieve the highest rewards and costs, having no penalty information [49]. Conversely, LPPO should show a significant trade-off between average cost and success or present a failure at maintaining the cost threshold when set to low values [25, 49]. In contrast, we expect the penalty-based methods to achieve promising returns while significantly reducing the cost and the number of violations during the training.

6.1 Robotic Navigation Tasks

We first introduce our safety navigation tasks that enable rapid testing of policies in realistic contexts. Our environments rely on Unity [19] as it offers rapid prototyping, Gym compatibility, and interface with the Robotic Operating System (ROS), which allows the transfer of the trained policies to actual robots.

In more detail, we employed navigation since it represents a well-known problem in model-free DRL [30, 64], prior safe DRL works [49], and multi-agent DRL [31, 32] literature. While standard navigation planners use a map of the environment and dense sensor information, DRL setups consider significantly more challenging conditions, such as not having a map and relying only on sparse local sensing. We use a similar encoding to prior work [30, 36, 57, 64]. Such encoding in this contest consists of 22 sparse laser scans with a limited range and two values for the target relative position (i.e., distance and heading), resulting in a vector $\mathbf{x} \in \mathbb{R}^2 4$ as input for the agent. Discrete actions encode angular and linear velocities to reduce training times while maintaining good navigation skills [30].[8] The agent thus obtains a dense reward given by the distance (d) difference (Δ) from the goal in two consecutive steps, with a per-step penalty ($-\eta$) that provides an incentive for shorter paths. Formally, at step t, the agent receives a reward:

[7]For a fair comparison, we set the cost threshold of LPPO to the average cost obtained by PPO_cost

[8]Our environments also support continuous actions and different domain randomization of the tasks and physical properties through the Unity editor.

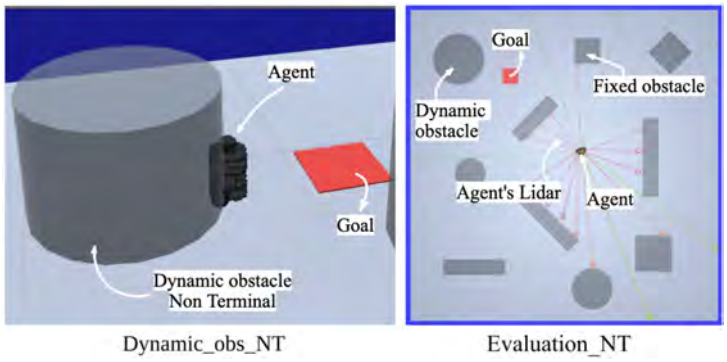


Fig. 9. Overview of the Dynamic and Evaluation tasks with different obstacles. Non-terminal obstacles allow the robot to cross them, experiencing more unsafe states. The evaluation environment has both fixed and dynamic non-terminal obstacles.

$$r_t = \begin{cases} 1 & \text{if goal reached} \\ \Delta(d_{t-1}, d_t) - \eta & \text{otherwise} \end{cases} \tag{6}$$

Each collision returns a positive cost signal that can be used to compute the desired penalty (e.g., violation), enabling a straightforward application for different penalty-based objectives (as in Equation 5) or constraints. We introduce two training and one testing environment with different obstacles, namely *Fixed_obs_NT*, *Dynamic_obs_NT*, and *Evaluation_NT* depicted in Figure 9. Such a variety of conditions provides different settings for properly evaluating safe DRL approaches. The environments inherit several characteristics from known benchmarks such as SafetyGym [49]. The proposed scenarios share a 4×4 squared meters size (6×6 squared meters for the testing one), randomly generated obstacle-free goals, and a timeout at 500 steps. The main features of the environments are the following:

- *Fixed_obs_NT* has fixed non-terminal (NT) obstacles. The environment returns a signal upon each collision that can be used to model cost functions or other penalties. Non-terminal obstacles are visible to the lidars, but non-tangible, i.e., the Turtlebot3 can pass through them. This class of obstacles represents the main challenge to designing safe DRL solutions, as the robot could obtain positive rewards by crossing an obstacle towards the goal at the expense of unsafe behaviors.
- *Dynamic_obs_NT* has cylindrical-shaped dynamic NT obstacles. Such obstacles move toward random positions at a constant velocity, representing a harder challenge. The obstacles can travel on the robot's goal, so the agent must learn a wider variety of behaviors (e.g., react to an approaching obstacle, stand still to wait for the goal to clear).
- *Evaluation_NT*: we use this evaluation environment to test the generalization abilities of trained policies to new situations. This scenario is wider and contains both fixed and dynamic non-terminal obstacles of different shapes.

6.2 Comparison Between hand-designed Properties and CROP

We perform a preliminary experiment showing that creating an exhaustive set of properties encompassing all unsafe behaviors in difficult environments is unfeasible. Hence, we show the benefits of our CROP method to collect and refine safety property at training time. In our evaluation, we consider the *Fixed_obs_NT* scenario using PPO_violation as a baseline algorithm and hand-design a set of 15 safety properties. The definitions of these specifications follow the rational, safe behaviors for navigation described in Section 3.3.1. Figure 10 shows

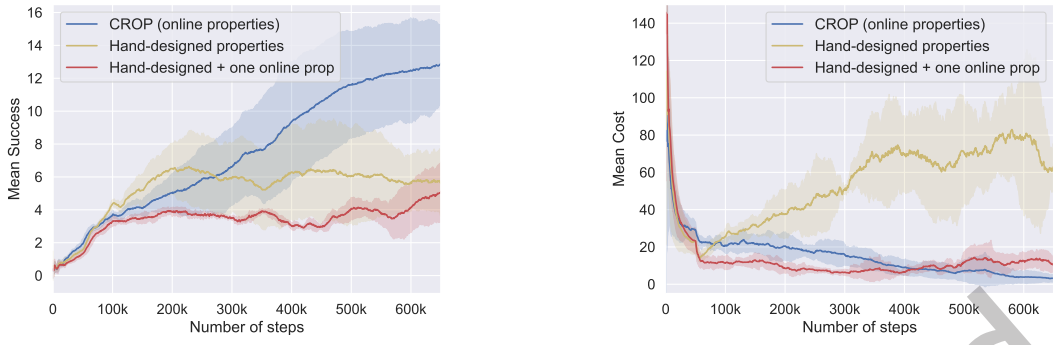


Fig. 10. Comparison between a PPO_violation method that uses: CROP (blue), hand-designed (yellow), and hand-designed plus one generated property upon collision (red) properties.

the average success (left) and cost (right) obtained by PPO_violation when using only: (i) properties generated during the training with CROP, (ii) hand-designed properties as described in Section 3.3.1, (iii) the setup proposed in [33] paper considering hand-designed properties plus one property generated online upon each collision. Note that the latter does not consider the property buffer and the refinement method proposed in CROP [39]. Clearly, computing the violation on CROP's properties provides more information to the learning agent on unsafe behaviors that are challenging to hand-design by a system designer. As such, PPO_violation using CROP achieves the highest performance in terms of average success while also converging to the lowest amount of collisions. For this reason, the following experiments only consider the PPO_violation approach that uses CROP. We will refer to this method as PPO_CROP.

Coverage Online Properties. Although is unfeasible to ensure comprehensive coverage of all safety requirements, to investigate the scope of safety requirements collected by CROP, we compare and report in Fig.11 the mean number of properties employed to compute the violation value (i.e., properties that contain the unsafe interaction in their domain-codomain) in the case of PPO_CROP (ours) and PPO_violation (which considers the set of hand-designed properties for navigation). As expected, CROP collects a high number of properties on the task,

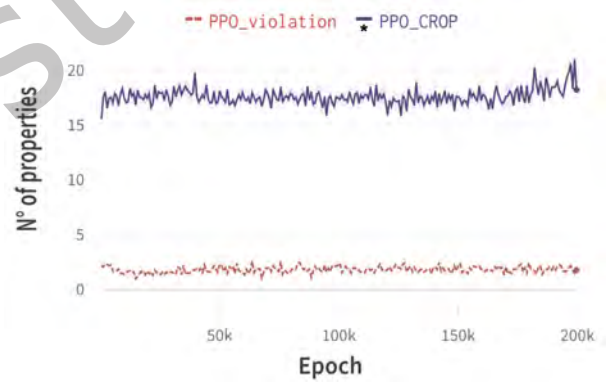


Fig. 11. Average number of proprieties' used for the *violation* computation during the training of PPO_violation and our PPO_CROP.

while the hand-designed approach cannot cope with the complexity of the environment and employs a reduced number of properties for computing the *violation*. Importantly, we note that the number of properties used in CROP tends to slightly increase during the training, while it remains constant for the other methodology. Such a result implies that collecting properties during the training by the agent allows it to find borderline cases of unsafe situations that are difficult to explore using hand-designed properties.

6.3 Ensuring the Markov Property

This section aims to tackle one possible drawback of the proposed approach. In particular, by employing the violation value as a penalty, the next state and reward could not be independent of past experienced states and actions, thus potentially violating the Markov property of the problem formulation. Nonetheless, by leveraging the intuitions of Foerster et al. [14], we can address such an issue by augmenting the state information with the parameters of the policy at the previous step. While it is unfeasible to include all the policies' weights due to their high dimensionality, we concatenate the output policy parameters, which fully describe the action distribution and the current training step. Previous work [14] refers to such augmentation as *fingerprint* and discusses how this information disambiguates trajectories in more detail. Hence, we adopt the same strategy to solve the issues the violation value could present with respect to the Markov property.

However, while having theoretically grounded drawbacks, practical implementations can often benefit from lower dimensional state representations. For this reason, we compare the theoretically justified version of PPO_CROP with the fingerprint, called PPO_CROP (markovian) and PPO_CROP that does not use it, in the *Fixed_obs_NT* task. Figure 12 confirms the practical advantages of maintaining a lower-dimensional input representation, as PPO_CROP (markovian) leads to lower performance than PPO_CROP in the training regime considered in our experiments. While PPO_CROP (markovian) may eventually converge to the same success and cost values as PPO_CROP, it remains much less sample efficient. Hence, the following sections will consider the PPO_CROP implementation.

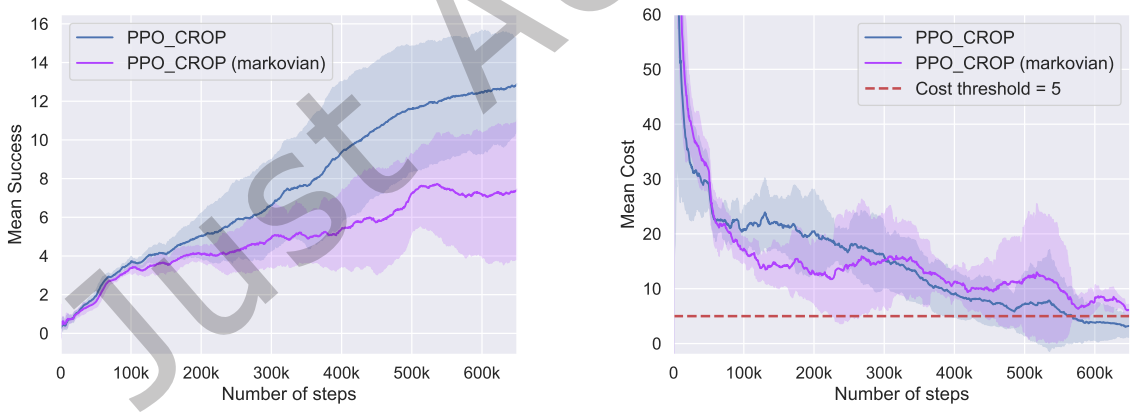


Fig. 12. Comparison between PPO_CROP and PPO_CROP (markovian) with fingerprint.

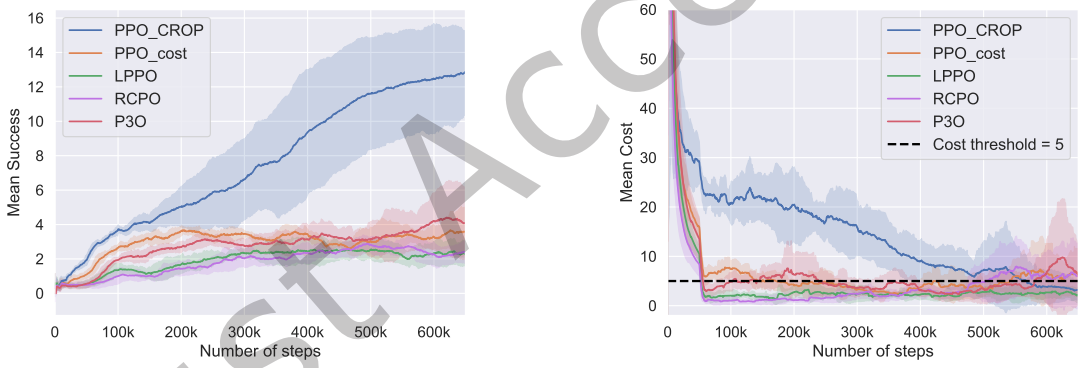
6.4 Empirical Evaluation on Navigation Tasks

We now compare the performance of our best-performing approach, PPO_CROP (i.e., with no hand-designed properties and state-augmentation), over the considered safe DRL baselines. Figures 13, 14 show the results of

our evaluation with *Fixed* and *Dynamic* obstacles for our PPO_CROP, PPO_cost (that uses the cost as a fixed penalty) and the baselines LPPO, RCPO, and P3O.

When considering fixed obstacles (Figure 13), our PPO_CROP outperforms other methodologies in terms of average success rate while significantly reducing the cost (i.e., number of collisions) and violation at convergence. Even though in lowering the mean cost and violation, PPO_CROP is slower than the other methodologies, at convergence, it shows a remarkable improvement. The nature of this result is clear: our approach requires experiencing different unsafe behaviors and thus collecting different safety properties before achieving in-depth navigation information. However, as the training steps and knowledge augment, our method outperforms the other methodologies, augmenting the mean success rate while lowering the mean cost. Moreover, by focusing on the violation metric, which, as stated before, measures the probability of the policy to commit an unsafe action, at convergence, PPO_CROP shows a $\approx 4.5\%$ lower violation over PPO_cost.

Considering the number of points used to compute the violation ($m = 10000$) and three common safety properties collected by all the approaches, such improvement maps to 1350 fewer collisions on average. Moreover, LPPO, RCPO, and P3O successfully maintain the cost accumulation under the specified threshold but fail to learn good navigation behaviors. PPO_CROP also achieves a $\approx 2.4\%$, $\approx 1.3\%$, and $\approx 3.6\%$ violation improvement, translating into 720, 390, and 1080 fewer collisions over RCPO, LPPO, and P3O at convergence, respectively. Crucially, the lower number of successes is not motivated by performing longer but safer paths, as PPO_CROP reaches a significantly higher number of successes while maintaining lower violations and cost values.

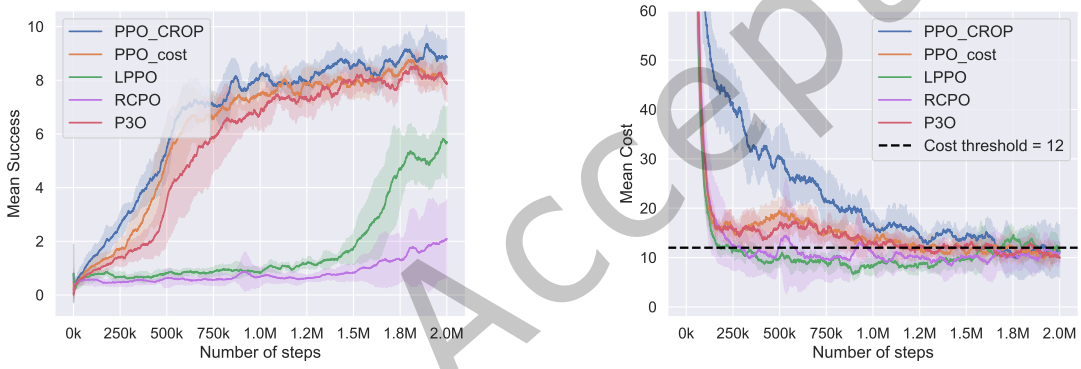


	Mean Violation		
	100k	**300k**	**650k**
*PPO_CROP	6.7 ± 1.0	4.8 ± 1.8	$\mathbf{2.2 \pm 1.2}$
LPPO	1.1 ± 0.5	2.1 ± 1.4	3.5 ± 1.2
RCPO	$\mathbf{0.37 \pm 0.23}$	$\mathbf{1.66 \pm 0.97}$	4.6 ± 5.3
PPO_cost	3.7 ± 1.0	2.5 ± 0.4	6.7 ± 5.8
P3O	2.4 ± 1.1	2.2 ± 0.8	5.8 ± 3.9

Fig. 13. Comparison between PPO_CROP, PPO_cost, RCPO, LPPO, and P3O in the *Fixed_obs_NT* environment. The y-axis limit of the mean cost plot has been set to 60 to better visualize the cost value at convergence. In the table, we report the mean violation value at three different global steps for each method tested.

In general, the low average success obtained by the Lagrangian baselines confirms the limitations of using indicator cost functions. We also remark on the lack of information carried by the cost since all the methods achieve similar average cost at convergence but result in significantly different violation values. This further motivates our use of the violation value to foster safety, as the cost could not be sufficient to determine how safely a policy behaves in continuous spaces. We reach similar conclusions in the task with dynamic obstacles, where our method achieves better or comparable successes and cost values over the counterparts but significantly reduces the violations during the training, showing an $\approx 1.9\%$, $\approx 0.9\%$, $\approx 2.00\%$, and $\approx 0.3\%$ violation improvement over LPPO, RCPO, PPO_cost, and P3O. This difference translates to 570, 270, 600, and 90 fewer collisions on average when employing PPO_CROP over LPPO, RCPO, PPO_cost, and P3O, respectively.

For a fair evaluation, all algorithms have been trained over the same parameters and configurations of dynamic obstacles (i.e., they experience the same random obstacle movements during the training phase, which change over different seeds). Additionally, we note that the additional value functions commonly required by existing safe baselines lead to a comparable wall-clock time overhead (considering the same hardware architecture) over the online safety properties collection and verification procedures of our method, with a maximum deterioration of 30% on the unsafe PPO's total training time.



	Mean Violation		
	500k	1M	2M
*PPO_CROP	16.2 ± 2.8	10.9 ± 1.7	$\mathbf{7.8 \pm 3.7}$
LPPO	$\mathbf{8.9 \pm 1.4}$	$\mathbf{7.1 \pm 1.4}$	9.7 ± 2.2
RCPO	9.0 ± 3.4	9.1 ± 4.3	8.7 ± 1.6
PPO_cost	15.8 ± 2.9	11.5 ± 1.1	9.8 ± 1.2
P3O	12.6 ± 0.4	10.8 ± 2.8	8.1 ± 2.3

Fig. 14. Comparison between PPO_CROP, PPO_cost, RCPO, LPPO and P3O on the *Dynamic_obs_NT* environment. The y-axis limit of the mean cost plot has been set to 60 to better visualize the cost value at convergence.

Formal Violation Value at Convergence. As discussed in Section 3.2, the violation value computed using the proposed Monte Carlo sampling-based method approximates the one computed with FV tools. However, considering the gap between the two values, performing an additional analysis using FV before deploying a trained model is crucial. This allows us to provably guarantee whether a model is safe or not with respect to desired safety

properties. We also leverage such analysis to confirm our claims on the superior performance of PPO_CROP in terms of violations.

Table 2. Mean violation expressed in percentage values computed using formal verification on the best model at convergence. The results of *Fixed obstacles* environment are reported on the left, while those of *Dynamic obstacles* are reported on the right.

	Method				
Property	**PPO_cost**	***PPO_CROP**	**RCPO**	**LPPO**	**P3O**
$\mathcal{P}_{\uparrow}$	2.1%	**1.12%**	1.7%	1.83%	1.26%
$\mathcal{P}_{\rightarrow}$	6%	1.81%	0.82%	**0.5%**	0.67%
$\mathcal{P}_{\leftarrow}$	0.7%	**0.1%**	0.89%	0.9%	1.4%
Mean	2,.93%	**1.01%**	1.14%	1.08%	1.1%

	Method				
Property	**PPO_cost**	***PPO_CROP**	**RCPO**	**LPPO**	**P3O**
$\mathcal{P}_{\uparrow}$	0.05%	0.04%	0.34%	0.134%	**0.02**%
$\mathcal{P}_{\rightarrow}$	0.04%	0.01%	0.03%	**0.005%**	0.06%
$\mathcal{P}_{\leftarrow}$	0.21%	**0.03%**	0.12%	0.35%	0.17%
Mean	0.1%	**0.03%**	0.16%	0.16%	0.08%

Table 2 shows the average formal violation value computed on the models at convergence in the *Fixed* and *Dynamic* tasks with respect to the hand-designed properties of Section 3.3.1. We consider hand-designed properties for a fair comparison since PPO_CROP is trained to minimize violations of the CROP's properties, but not these hand-designed ones. All methodologies achieve a violation value of a few percentage points, confirming the learning of basic safe navigation skills. Crucially, focusing on $\mathcal{P}_{\rightarrow}$, we notice that only for this specific property, LPPO is safer than PPO_CROP. This confirms our hypothesis regarding the significant trade-off between average cost and success that these cost-based methods are subject to. In fact, checking the behavior of LPPO for this property, we noticed that the policy chose a *"stay in place"* action, thus not raising the average cost but sacrificing mean success. In contrast, considering the mean violation on all three safety properties, PPO_CROP turns out to be the safest approach even when employing FV.[9]

Evaluating Trained Policies in an Unseen Scenario. To test the generalization skills of the trained policies in a previously unseen scenario, we perform an additional experiment on the *Evaluation_NT* task depicted on the right of Figure 9.

Table 3 reports each model's average success, cost, and (the approximate) violation at convergence. Results confirm the higher performance of PPO_CROP. In particular, our method shows superior navigation skills by achieving a higher number of successes while being safer than the cost counterparts, LPPO, RCPO, and P3O.

Table 3. Evaluation results in the *Evaluation_NT* environment

Method	**Mean Success**	**Mean Cost**	**Mean Violation**
*PPO_CROP	**8.8 ± 0.8**	**1.8 ± 0.9**	**1.3 ± 0.6**
LPPO	4.0 ± 0.9	2.1 ± 1.6	1.8 ± 1.3
RCPO	4.9 ± 1.2	2.1 ± 1.9	2.0 ± 1.5
PPO_cost	8.4 ± 1.5	2.4 ± 0.5	2.1 ± 0.5
P3O	6.3 ± 1.5	2.3 ± 0.2	1.9 ± 0.7

[9]We note results in Table 2 differ from the ones published in our previous contributions as we refined the size of the input areas for the three hand-designed properties. In particular, we employed stricter bounds by leveraging the information returned by CROP's online collected properties.

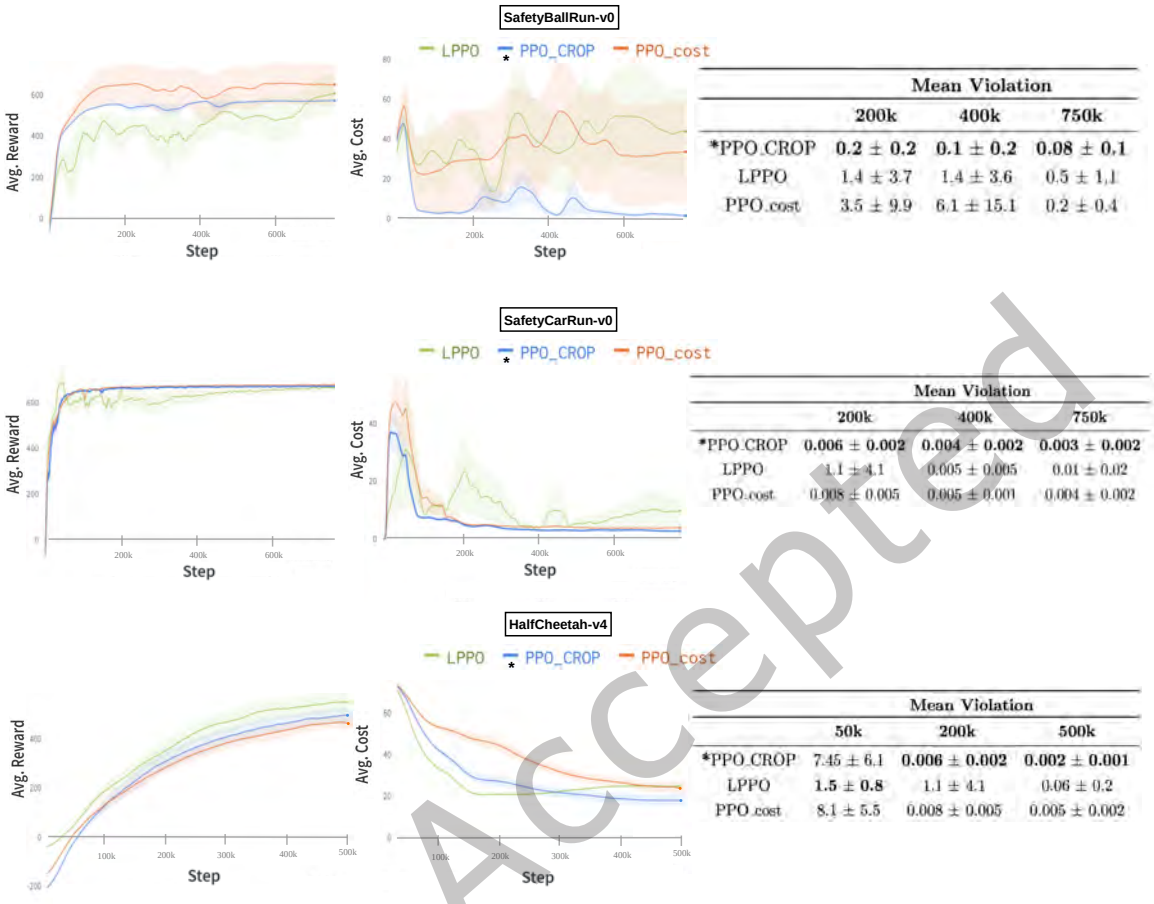
Transferring Policies to Real Robots. Considering the Unity game engine compatibility with ROS, it is possible to transfer policies trained in simulation on ROS-enabled platforms such as our Turtlebot3. Figure 15 shows an example of the same environment in simulation (left) and in the real world (right), on which we tested our trained policies. Our real-world experiments compare PPO_CROP and LPPO (which resulted in the overall safest approach after ours) in many real corner-case scenarios. The test on the real robot confirms that PPO_CROP shows superior performance with respect to LPPO. In particular, we confirm our hypotheses on the low expressiveness of cost functions. As shown in the results PPO_CROP, by exploiting the collection and refinement of safety properties in combination with the violation value metric leads the agent to acquire safer navigation skills in many corner cases that are difficult to code with cost functions or hand-designed safety specifications. Crucially, we noticed that when the agent trained with LPPO experiences unobstructed forward lidar perception, it tends to favor selecting the shorter path toward the goal, often resulting in collisions with obstacle corners. In contrast, the agent trained using our approach enables the policy to opt for a longer yet safer trajectory toward the goal.

Fig. 15. Overview between our Unity simulation environment (left) and the real-world scenario (right).

6.5 Empirical Evaluation in Standard Safety Benchmarks

We performed several additional experiments in the SafetyBallRun, SafetyCarRun, and SafetyHalfCheetahVelocity tasks, which are standard benchmarks employed by the safe DRL research community. These tasks consider different simulated robots and safety specifications, which allow showing the general effectiveness of using the violation value as a metric for the safety level. These experiments aim to further confirm the benefit of CROP and the use of the violation value as a penalty in a different domain known in the literature.[10] In more detail, we consider the same hyperparameters of our navigation experiments. However, we only consider LPPO as a Lagrangian baseline since it showed the best trade-off between performance and safety over the other baselines. The following results consider the average reward, cost, and violation collected over ten runs with different random seeds. Figure 16 shows the average performance obtained by our algorithm during the entire training phase. Crucially, these results confirm the behavior of previous works, where LPPO struggles to keep the cost threshold in some scenarios due to the task complexity. Moreover, LPPO converges to comparable performance with PPO_cost. In contrast, the PPO_CROP maintains the best trade-off between reward, cost, and violation through the training phases.

[10]We refer to the original works for more details about the environments and the cost functions used for the task[2], github.com/SvenGronauer/Bullet-Safety-Gym



	Mean Violation		
	200k	400k	750k
*PPO_CROP	**0.2 ± 0.2**	**0.1 ± 0.2**	**0.08 ± 0.1**
LPPO	1.4 ± 3.7	1.4 ± 3.6	0.5 ± 1.1
PPO_cost	3.5 ± 9.9	6.1 ± 15.1	0.2 ± 0.4

	Mean Violation		
	200k	400k	750k
*PPO_CROP	**0.006 ± 0.002**	**0.004 ± 0.002**	**0.003 ± 0.002**
LPPO	1.1 ± 4.1	0.005 ± 0.005	0.01 ± 0.02
PPO_cost	0.008 ± 0.005	0.005 ± 0.001	0.004 ± 0.002

	Mean Violation		
	50k	200k	500k
*PPO_CROP	7.45 ± 6.1	**0.006 ± 0.002**	**0.002 ± 0.001**
LPPO	**1.5 ± 0.8**	1.1 ± 4.1	0.06 ± 0.2
PPO_cost	8.1 ± 5.5	0.008 ± 0.005	0.005 ± 0.002

Fig. 16. Plots BulletSafetyGym envs

Figure 17 summarizes these results using the Pareto frontier of the average reward versus cost at convergence, which drastically improves (up and to the left) with PPO_CROP.

7 RELATED WORK

Garcıa and Fernández [15] presented an exhaustive taxonomy of the main families of approaches for safe DRL, analyzing the pros and cons of each category. At a high-level, safety in DRL has been pursued through formal verification techniques, control-theoretic approaches, model-based methods, constrained optimization, and penalty-based formulations. In this work, we focus on the model-free setup, where a variety of methods have been developed to promote safe exploration and policy learning. Nonetheless, in the following, we briefly mention the other research directions, discussing their benefits and limitations.

Formal Methods-based Safe DRL. A line of research at the intersection of DRL and formal methods guarantees safety by synthesizing a runtime enforcement mechanisms—commonly referred to as a *shields*—to ensure that the learned agent adheres to logic specifications [9, 13]. These methods provide strong safety

Fig. 17. Pareto frontiers of the methodologies tested in several BulletSafetyGym environments. The x-axis represents the average cost, while the y-axis the average reward at convergence. The best possible result is achieved by reaching the top left corner of the plot.

guarantees but struggle to scale due to the exponential growth in the size of the synthesized automaton. As a result, they are often confined to low-dimensional domains or discrete environments. Statistical verification (SV) techniques [63] have also been employed to validate that the agent's behavior satisfies signal temporal logic (STL) properties, which are closely related to the specifications used in formal verification. However, standard SV methods do not directly provide actionable penalties for constraint violations, unlike our proposed framework based on violation signals.

Control Barrier Function-based Safe DRL. Control-theoretic approaches such as control barrier functions (CBFs) offer a promising direction for ensuring safety, also in continuous control systems where shielding methods typically struggle. CBFs enforce forward invariance of a safe set by modifying control actions to stay within safe regions [42, 59]. Recent works have also extended CBFs to learning-based settings by combining them with formal methods and RL [17, 24, 54]. However, these approaches require access to accurate system dynamics or high-quality estimators, which limits their applicability in high-dimensional or partially observed settings.

Model-based Safe DRL. Model-based DRL methods have been investigated in constrained and unconstrained settings [20, 66], but having (or approximating) a model is not always possible. Moreover, Gaussian process (GP)-based methods model the uncertainty of the environment and utilize probabilistic safety guarantees to guide exploration [3, 68]. However, GPs scale poorly with data and typically assume smooth, low-dimensional dynamics, which again limits their general use in DRL scenarios. Furthermore, these methods often require querying the system extensively before executing policies, which can be impractical in safety-critical settings.

Constrained and Model-Free Policy Optimization. In contrast to model-based approaches, model-free constrained optimization methods have become a popular paradigm for safe DRL in single and multi-agent contexts [2, 6, 7, 62, 67]. Constrained policy optimization (CPO) [2] provides a principled framework to enforce constraints during policy updates, but it suffers from infeasibility issues due to approximation errors and relies on expensive second-order computations. Similarly, PCPO [62] uses second-order methods and has shown mixed improvements over CPO [67]. A different line of work considers Lyapunov-based approaches [11]. However, the cardinality of Lyapunov constraints equals the number of states. Hence, Lagrangian methods, described in Section 2, are commonly employed over these demanding solutions due to their simplicity and good performance. In general, constrained DRL presents significant drawbacks, as shared by all the approaches described above. For example, setting an incorrect threshold value may lead to algorithms being too permissive or, conversely, too restrictive [15]. Moreover, the guarantees of some of these approaches rely on strong assumptions that can not be satisfied in standard DRL settings, such as having access to an optimal policy. Constrained DRL is thus

not devoid of short-term fatal behaviors as it can fail at satisfying the safety constraints. Moreover, constraints naturally limit exploration, which may result in getting stuck in local optima or failing to learn desired behaviors properly [16, 27]. Despite the mentioned issues, constrained DRL methods are an important benchmark for safe DRL, and in this work, we decided to compare our approach with Lagrangian methods [55] as they reduce the complexity of prior approaches while showing higher performance and comparable constraints satisfaction.

Reward Shaping and Penalty-Based Methods. Another way to address safety in DRL is by retrain and incorporating penalties, which has been proven effective in several works [1, 15, 22, 29, 38, 53]. For example, IPO [25] uses a penalty function based on constraints, giving a zero penalty when constraints are satisfied and a negative infinity upon violation. However, tuning IPO's specific parameters and the constraint threshold is not straightforward and may result in lower performance over Lagrangian methods. Hence, RCPO [58] proposes a straightforward Lagrangian-based penalty and often leads to better performance. Similarly, P3O [65] uses the scaled gap between an additional cost value function and desired safety thresholds as a penalty to eliminate cost constraints and optimize for safety. Considering our goal of incorporating the violation value as a penalty, we decided to also compare our penalty-based approach with RCPO and P3O, which we believe are the closest related works to ours since they employed adaptive penalties.

8 DISCUSSION AND FUTURE DIRECTIONS

This paper introduced an unconstrained DRL framework that leverages local violations of input-output conditions to foster safety. In detail, we discussed the limitations of using cost functions as in safe DRL [15] presenting: (i) the CROP framework that allows the collection and refinement of safety properties during training, thus overcoming the limitations of hand-designed approaches; (ii) a sample-based approach to approximate a *violation* metric and use it as a penalty in DRL algorithms. We provided provable probabilistic guarantees on the correctness of our approximation, showing that such a violation introduces task-level safety specifications into the optimization, addressing the potential lack of safety information of cost-based approaches. We empirically demonstrated that CROP allows us to obtain a more robust approach with respect to other safe DRL methodologies, promoting safer behaviors while maintaining similar or better returns.

This work paves the way for several research directions, including extending our task suite to create a general safe DRL benchmark. While our experiments focus on robotic navigation and locomotion tasks, the proposed method is domain-agnostic and could be potentially applied to other areas such as healthcare and autonomous driving. Future work could study the applicability of our pipeline to these challenging scenarios. Moreover, it would also be interesting to exploit the online safety properties collected using CROP to design a shield to avoid unsafe behaviors at deployment or to retrain the agent on those specific unsafe regions. Finally, our insights on violation could be used to model desired behaviors in single and multi-agent applications to improve performance and sample efficiency.

ACKNOWLEDGMENTS

The work of Luca Marzari, Ferdinando Cicalese, and Alessandro Farinelli was partially funded by the PNRR MUR project PE0000013-FAIR. The work of Christopher Amato was partially funded by the Army Research Office under award number W911NF20-1-0265.

REFERENCES

[1] David Abel, Will Dabney, Anna Harutyunyan, Mark K. Ho, Michael L. Littman, Doina Precup, and Satinder Singh. 2021. On the Expressivity of Markov Reward. In *Conference on Neural Information Processing Systems (NeurIPS)*.

[2] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. 2017. Constrained Policy Optimization. In *International Conference on Machine Learning (ICML)*.

[3] Filippo Airaldi, Bart De Schutter, and Azita Dabiri. 2023. Learning safety in model-based Reinforcement Learning using MPC and Gaussian Processes. *IFAC* 56, 2 (2023), 5759–5764.

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

[5] Guy Amir, Davide Corsi, Raz Yerushalmi, Luca Marzari, David Harel, Alessandro Farinelli, and Guy Katz. 2023. Verifying learning-based robotic navigation systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 607–627.

[6] 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.

[7] Ayhan Alp Aydeniz, Enrico Marchesini, Robert Loftin, Christopher Amato, and Kagan Tumer. 2024. Safe Multiagent Coordination via Entropic Exploration. https://arxiv.org/abs/2412.20361

[8] Christopher Brix, Mark Niklas Müller, Stanley Bak, Taylor T Johnson, and Changliu Liu. 2023. First three years of the international verification of neural networks competition (VNN-COMP). *International Journal on Software Tools for Technology Transfer* 25, 3 (2023), 329–339.

[9] Steven Carr, Nils Jansen, Sebastian Junges, and Ufuk Topcu. 2022. Safe Reinforcement Learning via Shielding for POMDPs. In *arXiv*.

[10] Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. 2018. A Lyapunov-based Approach to Safe Reinforcement Learning. In *Conference on Neural Information Processing Systems (NeurIPS)*.

[11] Yinlam Chow, Ofir Nachum, Aleksandra Faust, Mohammad Ghavamzadeh, and Edgar A. Duéñez-Guzmán. 2019. Lyapunov-based Safe Policy Optimization for Continuous Control. In *International Conference on Machine Learning (ICML)*.

[12] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. 2019. A hitchhiker's guide to statistical comparisons of reinforcement learning algorithms. *arXiv preprint arXiv:1904.06979* (2019).

[13] Ingy Elsayed-Aly, Suda Bharadwaj, Christopher Amato, Rüdiger Ehlers, Ufuk Topcu, and Lu Feng. 2021. Safe Multi-Agent Reinforcement Learning via Shielding. In *AAMAS*.

[14] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip H. S. Torr, Pushmeet Kohli, and Shimon Whiteson. 2017. Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning*.

[15] Javier Garcıa and Fernando Fernández. 2015. A comprehensive survey on safe reinforcement learning. In *Journal of Machine Learning Research (JMLR)*.

[16] Zhang-Wei Hong, Tzu-Yun Shann, Shih-Yang Su, Yi-Hsiang Chang, and Chun-Yi Lee. 2018. Diversity-Driven Exploration Strategy for Deep Reinforcement Learning. In *Conference on Neural Information Processing Systems (NeurIPS)*.

[17] Hanjiang Hu, Yujie Yang, Tianhao Wei, and Changliu Liu. 2024. Verification of neural control barrier functions with symbolic derivative bounds propagation. *arXiv preprint arXiv:2410.16281* (2024).

[18] Andrew Ilyas, Logan Engstrom, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. 2018. A Closer Look at Deep Policy Gradients. In *ICLR*.

[19] 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*.

[20] Parv Kapoor, Anand Balakrishnan, and Jyotirmoy V. Deshmukh. 2020. Model-based Reinforcement Learning from Signal Temporal Logic Specifications. In *arXiv*.

[21] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International conference on computer aided verification*. Springer, 97–117.

[22] Zachary C. Lipton, Jianfeng Gao, Lihong Li, Jianshu Chen, and Li Deng. 2016. Combating Reinforcement Learning's Sisyphean Curse with Intrinsic Fear. In *arXiv*.

[23] 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.

[24] Simin Liu, Changliu Liu, and John Dolan. 2023. Safe control under input limits with neural control barrier functions. In *Conference on Robot Learning*. PMLR, 1970–1980.

[25] Yongshuai Liu, Jiaxin Ding, and Xin Liu. 2020. IPO: Interior-point Policy Optimization under Constraints. In *AAAI Conference on Artificial Intelligence*.

[26] Yongshuai Liu, Avishai Halev, and Xin Liu. 2021. Policy Learning with Constraints in Model-free Reinforcement Learning: A Survey. In *International Joint Conference on Artificial Intelligence (IJCAI)*. 4508–4515.

[27] Enrico Marchesini and Christopher Amato. 2022. Safety-Informed Mutations for Evolutionary Deep Reinforcement Learning. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 1966–1970.

[28] Enrico Marchesini and Christopher Amato. 2023. Improving Deep Policy Gradients with Value Function Search. In *ICLR*.

[29] Enrico Marchesini, Benjamin Donnot, Constance Crozier, Ian Dytham, Christian Merz, Lars Schewe, Nico Westerbeck, Cathy Wu, Antoine Marot, and Priya L. Donti. 2025. RL2Grid: Benchmarking Reinforcement Learning in Power Grid Operations. arXiv:2503.23101 [cs.LG] https://arxiv.org/abs/2503.23101

[30] E. Marchesini and A. Farinelli. 2020. Discrete Deep Reinforcement Learning for Mapless Navigation. In *ICRA*.
[31] 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. doi:10.1109/IROS51168.2021.9636349
[32] E. Marchesini and A. 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. doi:10.1109/ICRA46639.2022.9812341
[33] 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.
[34] Luca Marzari, Ferdinando Cicalese, and Alessandro Farinelli. 2025. Probabilistically Tightened Linear Relaxation-based Perturbation Analysis for Neural Network Verification. arXiv:2507.05405 https://arxiv.org/abs/2507.05405
[35] Luca Marzari, Davide Corsi, Ferdinando Cicalese, and Alessandro Farinelli. 2023. The #DNN-Verification Problem: Counting Unsafe Inputs for Deep Neural Networks. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*. 217–224.
[36] Luca Marzari, Davide Corsi, Enrico Marchesini, and Alessandro Farinelli. 2022. Curriculum learning for safe mapless navigation. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*. 766–769.
[37] Luca Marzari, Davide Corsi, Enrico Marchesini, Alessandro Farinelli, and Ferdinando Cicalese. 2024. Enumerating safe regions in deep neural networks with provable probabilistic guarantees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 21387–21394.
[38] Luca Marzari, Priya L. Donti, Changliu Liu, and Enrico Marchesini. 2025. Improving Policy Optimization via ε-Retrain. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*. 1464–1472.
[39] 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)*. IEEE, 7133–7139.
[40] Luca Marzari, Isabella Mastroeni, and Alessandro Farinelli. 2025. Advancing Neural Network Verification through Hierarchical Safety Abstract Interpretation. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*.
[41] Luca Marzari, Ameya Pore, Diego Dall'Alba, Gerardo Aragon-Camarasa, Alessandro Farinelli, and Paolo Fiorini. 2021. Towards Hierarchical Task Decomposition using Deep Reinforcement Learning for Pick and Place Subtasks. In *2021 20th International Conference on Advanced Robotics (ICAR)*. IEEE, 640–645.
[42] Luca Marzari, Francesco Trotti, Enrico Marchesini, and Alessandro Farinelli. 2025. Designing Control Barrier Function via Probabilistic Enumeration for Safe Reinforcement Learning Navigation. *IEEE Robotics and Automation Letters* 10, 10 (2025), 9630–9637. doi:10.1109/LRA.2025.3596431
[43] Mark Huasong Meng, Guangdong Bai, Sin Gee Teo, Zhe Hou, Yan Xiao, Yun Lin, and Jin Song Dong. 2022. Adversarial robustness of deep neural networks: A survey from a formal verification perspective. *IEEE Transactions on Dependable and Secure Computing* (2022).
[44] M. Mitzenmacher and E. Upfal. 2017. *Probability and Computing: Randomized and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge University Press.
[45] Ramon E Moore. 1962. *Interval arithmetic and automatic error analysis in digital computing*. Technical Report. Stanford Univ Calif Applied Mathematics And Statistics Labs.
[46] Andrew Y Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, Vol. 99. 278–287.
[47] J. Nocedal and S. Wright. 2006. *Numerical Optimization* (2 ed.). Springer.
[48] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. 2017. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087* (2017).
[49] Alex Ray, Joshua Achiam, and Dario Amodei. 2019. Benchmarking Safe Exploration in Deep Reinforcement Learning. In *arXiv*.
[50] Julien Roy, Roger Girgis, Joshua Romoff, Pierre-Luc Bacon, and Christopher Pal. 2022. Direct Behavior Specification via Constrained Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*.
[51] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv* (2017).
[52] Zhouxing Shi, Huan Zhang, Kai-Wei Chang, Minlie Huang, and Cho-Jui Hsieh. 2020. Robustness Verification for Transformers. In *International Conference on Learning Representations*.
[53] David Silver, Satinder Singh, Doina Precup, and Richard S. Sutton. 2021. Reward is enough. *Artificial Intelligence* 299 (2021), 103535.
[54] Oswin So, Zachary Serlin, Makai Mann, Jake Gonzales, Kwesi Rutledge, Nicholas Roy, and Chuchu Fan. 2024. How to train your neural control barrier function: Learning safety filters for complex input-constrained systems. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 11532–11539.

[55] Adam Stooke, Joshua Achiam, and Pieter Abbeel. 2020. Responsive Safety in Reinforcement Learning by PID Lagrangian Methods. In *International Conference on Machine Learning (ICML)*.
[56] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).
[57] L. Tai, G. Paolo, and M. Liu. 2017. Virtual-to-real DRL: Continuous control of mobile robots for mapless navigation. In *IROS*.
[58] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. 2019. Reward Constrained Policy Optimization. In *International Conference on Learning Representations*.
[59] Cumhur Erkan Tuncali, James Kapinski, Hisahiro Ito, and Jyotirmoy V. Deshmukh. 2018. Reasoning about Safety of Learning-Enabled Components in Autonomous Cyber-physical Systems. In *Design Automation Conference (DAC)*.
[60] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. 2016. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning (ICML)*. 1995–2003.
[61] Tianhao Wei, Hanjiang Hu, Luca Marzari, Kai S. Yun, Peizhi Niu, Xusheng Luo, and Changliu Liu. 2025. ModelVerification.jl: A Comprehensive Toolbox for Formally Verifying Deep Neural Networks. In *Computer Aided Verification*. 395–408.
[62] Tsung-Yen Yang, Justinian Rosca, Karthik Narasimhan, and Peter J. Ramadge. 2020. Projection-Based Constrained Policy Optimization. In *International Conference on Learning Representations (ICLR)*.
[63] Mojtaba Zarei, Yu Wang, and Miroslav Pajic. 2020. Statistical Verification of Learning-Based Cyber-Physical Systems. In *International Conference on Hybrid Systems: Computation and Control*.
[64] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard. 2017. Deep reinforcement learning with successor features for navigation across similar environments. In *IROS*.
[65] Linrui Zhang, Li Shen, Long Yang, Shixiang Chen, Bo Yuan, Xueqian Wang, and Dacheng Tao. 2022. Penalized Proximal Policy Optimization for Safe Reinforcement Learning. In *IJCAI*.
[66] Xinglong Zhang, Yaoqian Peng, Biao Luo, Wei Pan, Xin Xu, and Haibin Xie. 2021. Model-Based Safe Reinforcement Learning with Time-Varying State and Control Constraints: An Application to Intelligent Vehicles. In *arXiv*.
[67] Yiming Zhang, Quan Vuong, and Keith W. Ross. 2020. First Order Constrained Optimization in Policy Space. In *Conference on Neural Information Processing Systems (NeurIPS)*.
[68] Weiye Zhao, Tairan He, and Changliu Liu. 2023. Probabilistic Safeguard for Reinforcement Learning Using Safety Index Guided Gaussian Process Models. In *L4DC*.

Received 23 September 2024; revised 14 April 2025; accepted 4 September 2025