Safe Reinforcement Learning via Probabilistic Logic Shields

Wen-Chi Yang[1] , **Giuseppe Marra**[1] , **Gavin Rens**[2] and **Luc De Raedt**[1,3]
[1]Leuven AI, KU Leuven, Belgium
[2]Stellenbosch University, South Africa
[3]Centre for Applied Autonomous Sensor Systems, Örebro University, Sweden
{wenchi.yang, giuseppe.marra, luc.deradet}@kuleuven.be, gavinrens@sun.ac.za

Abstract

Safe Reinforcement learning (Safe RL) aims at learning optimal policies while staying safe. A popular solution to Safe RL is shielding, which uses a logical safety specification to prevent an RL agent from taking unsafe actions. However, traditional shielding techniques are difficult to integrate with continuous, end-to-end deep RL methods. To this end, we introduce Probabilistic Logic Policy Gradient (PLPG). PLPG is a model-based Safe RL technique that uses probabilistic logic programming to model logical safety constraints as differentiable functions. Therefore, PLPG can be seamlessly applied to any policy gradient algorithm while still providing the same convergence guarantees. In our experiments, we show that PLPG learns safer and more rewarding policies compared to other state-of-the-art shielding techniques.

1 Introduction

Shielding is a popular technique in Safe Reinforcement Learning (Safe RL) that aims to find an optimal policy while ensuring the learning agent's safety [Jansen *et al.*, 2020]. It relies on a logical component called a *shield* to monitor the agent's actions and rejects those that violate the given safety constraints. These *rejection-based* shields typically use formal verification, offering stronger safety guarantees than other safe exploration techniques [García and Fernández, 2015]. While early shielding techniques operate completely on symbolic state spaces [Jansen *et al.*, 2020; Alshiekh *et al.*, 2018; Bastani *et al.*, 2018], more recent approaches have incorporated neural policy learners to handle continuous state spaces [Hunt *et al.*, 2021; Anderson *et al.*, 2020; Harris and Schaub, 2020]. In this paper, we will also focus *on integrating shielding with neural policy learners.*

In current shielding approaches, the shields are deterministic, assuming that an action is either safe or unsafe in a particular state. However, this is an unrealistic assumption as the world is inherently uncertain, and safety is often a matter of degree rather than an absolute concept. For example, consider the scenario depicted in Fig. 1 where a car must detect obstacles from visual input, but the sensor readings are noisy. Such uncertainty arising from noisy sensors cannot be directly handled by rejection-based shields, as they often assume that agents have perfect sensors [Giacobbe *et al.*, 2021; Hunt *et al.*, 2021], which is unrealistic. Therefore, by working with probabilistic shields, *we will be able to better cope with such uncertainties and risks*.

Additionally, rejection-based shielding approaches may fail to learn an optimal policy even when provided with perfect safety information in all states [Ray *et al.*, 2019; Hunt *et al.*, 2021; Anderson *et al.*, 2020]. This is because the learning agent does not consider the rejected actions while updating its policy, assuming that all safe actions were sampled directly from its safety-agnostic policy instead of through the shield. By eliminating the mismatch between the shield and the policy, *we will be able to ensure convergence towards an optimal policy if one exists*.

We introduce *probabilistic shields* as an alternative to deterministic rejection-based shields. They produce a safer policy by incorporating noisy sensor readings into the original policy, as shown in Fig. 3 (right). By explicitly linking noisy sensor readings to probabilistic semantics, probabilistic shields can adjust the policy in proportion to the probabilistic safety. This approach also allows for shielding to be applied at the level of the policy instead of at the level of individual actions, which is typically done in the literature [Hunt *et al.*, 2021; Jansen *et al.*, 2020].

We propose the concept of *Probabilistic Logic Shields (PLS)* and its implementation in probabilistic logic programs. Probabilistic logic shields are probabilistic shields that model safety using logic. The safety specification is expressed as background knowledge, and its interaction with the learning agent and the noisy sensors is encoded in a probabilistic logic program, which is an elegant and effective way of defining a shield. Furthermore, probabilistic logic shields can be automatically compiled into a differentiable structure, allowing for the optimization of a single loss function through the shield, enforcing safety directly in the policy. Probabilistic logic shields have several benefits:

- *Realistic Safety Function*. Probabilistic logic shields use a probabilistic evaluation of safety, allowing for risk control.
- *Simpler Model.* Probabilistic logic shields use a simpler safety model that only represents internal safety-related properties, which is less demanding than many model-based approaches that require the full MDP [Jansen *et*

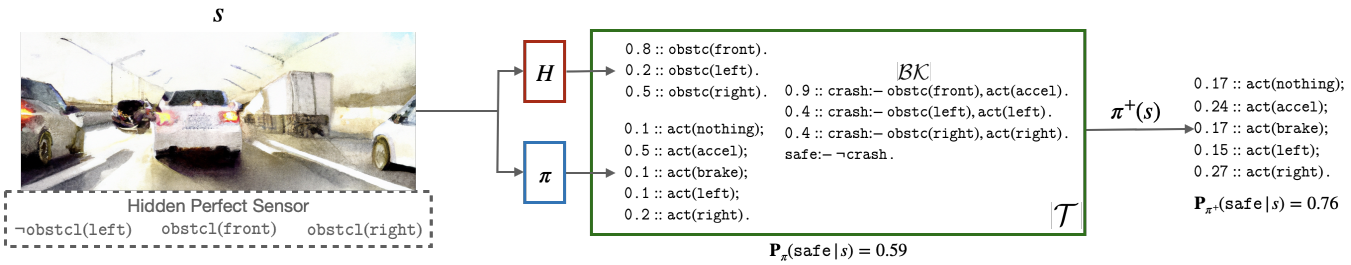

Figure 1: A motivating example of Probabilistic Logic Shields. We encode the interaction between the base policy π, the noisy sensors H and the safety specification $\mathcal{BK}$ using a ProbLog program $\mathcal{T}$. This provides a uniform language to express many aspects of the shielding process. The shielded policy $\pi^+(s)$ decreases the probability of unsafe actions, e.g. acceleration, and increases the likelihood of being safe.

al., 2020; Hunt *et al.*, 2021; Carr *et al.*, 2022].

- *End-to-end Deep RL.* Probabilistic logic shields are differentiable and can be seamlessly applied to any model-free RL agent such as PPO [Schulman *et al.*, 2016], TRPO [Schulman *et al.*, 2015], A2C [Mnih *et al.*, 2016], etc.
- *Convergence.* Probabilistic logic shields in deep RL provide convergence guarantees unlike rejection-based shields. More details can be found in Section 5.

2 Preliminaries

2.1 Probabilistic Logic Programming

We will introduce probabilistic logic programming (PLP) using the ProbLog syntax [De Raedt and Kimmig, 2015]. An *atom* is a predicate symbol followed by a tuple of logical variables and/or constants. A ProbLog theory (or program) $\mathcal{T}$ consists of a finite set of probabilistic facts $\mathcal{F}$ and a finite set of clauses $\mathcal{BK}$. A *fact* $\mathtt{f_i}$ is an expression denoting an atomic event. A *probabilistic fact* is an expression $\mathtt{p_i} :: \mathtt{f_i}$ where $\mathtt{p_i} \in [0, 1]$ denotes the probability of $\mathtt{f_i}$ being true, e.g. $0.8 :: \mathtt{obstc(front)}$ states that the probability of having an obstacle in front is 0.8. It is assumed that facts are independent of one another and the dependencies are specified by clauses (or rules). A *clause* is a universally quantified expression $\mathtt{h} :\!- \mathtt{b_1}, ..., \mathtt{b_n}$ where $\mathtt{h}$ is an atom and $\mathtt{b_1}, ..., \mathtt{b_n}$ is a conjunction of atoms or negated atoms, stating that $\mathtt{h}$ is true if all $\mathtt{b_i}$ are true. The clause defining $\mathtt{safe}$ in Fig. 1 states that it is safe when there is no crash. Each truth value assignment of all probabilistic facts $\mathcal{F} \subseteq \mathcal{T}$, denoted by $\mathcal{F}_k$, induces a possible world w_k where all ground facts in w_k are *true* and all that are not in w_k are *false*. Formally, the probability of a possible world w_k is defined as follows.

$$P(w_k) = \prod_{\mathtt{f_i} \in w_k} \mathtt{p_i} \prod_{\mathtt{f_i} \notin w_k} (1 - \mathtt{p_i})$$

An *annotated disjunction* (AD) is a clause with multiple heads $\mathtt{p_1} :: \mathtt{h_1}; \cdots; \mathtt{p_m} :: \mathtt{h_m}$ where each head $\mathtt{h_i}$ is mutually exclusive to one another, meaning that exactly one head is true when the body is true. The choice of the head is governed by a probability distribution and $\sum_{i=1}^{m} \mathtt{p_i} \leq 1$ [De Raedt and Kimmig, 2015; Fierens *et al.*, 2015]. E.g., $\{0.1 :: \mathtt{act(nothing)}; 0.5 :: \mathtt{act(accel)}; 0.1 :: \mathtt{act(brake)}; 0.1 :: \mathtt{act(left)}; 0.2 :: \mathtt{act(right)}\}$ is an AD (with no conditions), stating the probability distribution of selecting an action. In ProbLog's inference, ADs are internally converted to a set of probabilistic facts and clauses. The *success probability* of an atom q is the sum of the probabilities of all possible worlds that entail q. Formally, it is defined as $P(q) := \sum_{w_k \models q} P(w_k)$. Given a set of atoms E as evidence, the *conditional probability* of a query q is $P(q|E) = \frac{P(q,E)}{P(E)}$. For instance, in Fig. 1, the probability of being safe in the next state given that the agent accelerates is $P(\mathtt{safe}|\mathtt{act(accel)}) = \frac{0.14}{0.5} = 0.28$.

2.2 Markov Decision Process

A Markov decision process (MDP) is a tuple $M = \langle S, A, T, R, \gamma \rangle$ where S and A are a set of states and actions, respectively. $T(s, a, s') = P(s'|s, a)$ defines the probability of being in s' after executing a in s. $R(s, a, s')$ defines the immediate reward of executing a in s resulting in s'. $\gamma \in [0, 1]$ is the discount factor. A policy $\pi : S \times A \to [0, 1]$ defines the probability of taking an action in a state. $\pi(a|s)$ denotes the probability of taking action a in state s under policy π and $\pi(s)$ denotes the probability distribution over all actions in s.

2.3 Shielding

A shield is a reactive system that guarantees the safety of the learning agent by constraining its exploration, allowing it to only take actions that satisfy the given safety specification at run time [Jansen *et al.*, 2020]. A safety specification is usually defined by a Markov model and a temporal logic formula. For instance, when there is a car driving in front and the agent proposes to *accelerate*, a rejection-based shield (Fig. 3, left) will predict a crash and reject the action.

2.4 Policy Gradient

The objective of a RL agent is to find a policy π_θ (parameterized by θ) that maximizes the total expected return along the trajectory, formally,

$$J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})]$$

$J(\theta)$ is assumed to be finite for all policies. An important class of RL algorithms, particularly relevant in the setting of shielding, is based on *policy gradients*, which maximize $J(\theta)$

by repeatedly estimating the gradient $\nabla_\theta J(\theta)$. The general form of policy gradient methods is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(s_t, a_t)] \quad (1)$$

where Ψ_t is an empirical expectation of the return [Schulman *et al.*, 2016]. The expected value is usually computed using Monte Carlo methods, requiring efficient sampling from π_θ.

3 Probabilistic Shields

We assume that a probabilistic safety model is given to compute the probability that executing action a in state s is safe, i.e. $\mathbf{P}(\texttt{safe}|a, s)$. We use bold $\mathbf{P}$ to distinguish it from the underlying probability distributions $P(s'|s, a)$ of the MDP. The probabilistic safety model $\mathbf{P}(\cdot)$ needs to internally represent the safe-relevant dynamics, but it does not require the full representation of the MDP. Therefore, it is limited to safety-related properties and cannot be used to predict the next state s'. Rejection-based shields implement a special case of $\mathbf{P}$ that assumes that executing an action is either safe or unsafe, i.e. $\mathbf{P}(\texttt{safe}|s, a) \in \{0, 1\}$.

By marginalizing out actions based on their probabilities under π, the probability model $\mathbf{P}_\pi$ measures the likelihood of taking a safe action in s according to π.

$$\mathbf{P}_\pi(\texttt{safe}|s) = \sum_{a \in A} \mathbf{P}(\texttt{safe}|s, a)\pi(a|s) \quad (2)$$

Definition 3.1. ***Probabilistic Shielding*** *Given a base policy π and a probabilistic safety model $\mathbf{P}(\texttt{safe}|s, a)$, the shielded policy is*

$$\pi^+(a|s) = \mathbf{P}_\pi(a|s, \texttt{safe}) = \frac{\mathbf{P}(\texttt{safe}|s, a)}{\mathbf{P}_\pi(\texttt{safe}|s)}\pi(a|s) \quad (3)$$

Intuitively, a shielded policy π^+ re-normalizes the base policy π by increasing (resp. decreasing) the probabilities of the actions that are safer (resp. less safe) than average.

Proposition 1. *A shielded policy is always safer than its base policy in all states, i.e. $P_{\pi^+}(\texttt{safe}|s) \geq P_\pi(\texttt{safe}|s)$ for all s and π.*

Proof.

$$\begin{aligned}
&\mathbf{P}_{\pi^+}(\texttt{safe}|s) && \triangleright \text{Eq. (2)}\\
=&\sum_{a \in A} \pi^+(a|s)\mathbf{P}(\texttt{safe}|s, a) && \triangleright \text{Eq. (3)}\\
=&\frac{1}{\mathbf{P}_\pi(\texttt{safe}|s)}[\sum_{a \in A} \pi(a|s)\mathbf{P}(\texttt{safe}|s, a)^2] \\
& && \triangleright \text{Jensen's inequality}\\
\geq&\frac{1}{\mathbf{P}_\pi(\texttt{safe}|s)}[\sum_{a \in A} \pi(a|s)\mathbf{P}(\texttt{safe}|s, a)]^2 && \triangleright \text{Eq. (2)}\\
=&\mathbf{P}_\pi(\texttt{safe}|s)
\end{aligned}$$

The inequality comes from Jensen's inequality [Pishro-Nik, 2014], which states that $\mathbb{E}[g(X)] \geq g(\mathbb{E}[X])$ for any convex function $g(X)$. In this proof, $g(X) = X^2$. □

4 Probabilistic Logic Shields

In this paper, we focus on probabilistic shields implemented through *probabilistic logic programs*. The ProbLog program defining probabilistic safety consists of three parts, illustrated by Fig. 1.

The first component of the program is an annotated disjunction $\mathbf{AD}_s$ representing the policy $\pi(s)$ for state s. For example, in Fig. 1, $\pi(s)$ is $\mathbf{AD}_s = \{\texttt{0.1 :: act(nothing)}; \texttt{0.5 :: act(accel)}; \texttt{0.1 :: act(brake)}; \texttt{0.1 :: act(left)}; \texttt{0.2 :: act(right)}\}$.

The second component of the program is a set of probabilistic facts $\mathbf{F}_s$ representing an abstraction of the current state s. Such abstraction should contain information needed to reason about the safety of actions in s, and not a representation of the entire state. For example, in Fig. 1, the abstraction is represented as $\mathbf{F}_s = \{$ `0.8 :: obstc(front).` `0.2 :: obstc(left).` `0.5 :: obstc(right).`$\}$

The third component of the program is a set of clauses $\mathcal{BK}$ representing safety-related knowledge. For example, in Fig. 1, `0.9 :: crash :- obstc(front), act(accel)` states that the probability of having a crash is 0.9 if the agent accelerates when there is an obstacle in front of it. `safe :- \+crash` states that it is safe if no crash occurs.

Therefore, we obtain a ProbLog program $\mathcal{T}(s) = \mathcal{BK} \cup \mathbf{AD}_s \cup \boldsymbol{F}_s$, inducing a probability measure $\mathbf{P}_\mathcal{T}$. By querying this program, we can reason about safety of policies and actions. More specifically, we can use the same identical ProbLog program $\mathcal{T}(s)$ to obtain three distributions:

- action safety in s: $\mathbf{P}(\texttt{safe}|s, a) = \mathbf{P}_\mathcal{T}(\texttt{safe}|a)$
- policy safety in s: $\mathbf{P}_\pi(\texttt{safe}|s) = \mathbf{P}_\mathcal{T}(\texttt{safe})$
- the shielded policy in s: $\mathbf{P}_\pi(a|s, \texttt{safe}) = \mathbf{P}_\mathcal{T}(a|\texttt{safe})$

It is important to note that the safety model in ProbLog is an abstraction that may use a different representation than that of the underlying MDP.

Perception Through Neural Predicates We rely on two neural networks π and H to compute the probabilities $\mathbf{AD}_s$ and $\mathbf{F}_s$, which depend on the current state s. These networks take image input s and output the probabilities of the actions $\mathbf{AD}_s$ and the safe-relevant abstraction of the state $\mathbf{F}_s$, respectively. As depicted in Fig. 1, state s is fed into both networks, and the probabilities are then used as inputs to the ProbLog program. This feature is closely related to the notion of *neural predicate*, which can — for the purposes of this paper — be regarded as neural networks encapsulated inside logic. Since ProbLog programs are differentiable with respect to the probabilities in $\mathbf{AD}_s$ and $\mathbf{F}_s$, the gradients can be seamlessly backpropagated to network parameters during learning.

Leveraging Probabilistic Logic Programming Using ProbLog to implement the shield is convenient since it is a Turing-complete programming language that extends ProLog. ProbLog programs are differentiable, making them compatible with neural network optimization. Probabilistic logic programs can be compiled into arithmetic circuits using knowledge compilation [Darwiche, 2003]. Although the size of the circuit is generally exponential to the numbers of facts, ADs and clauses, once the circuit is obtained, inference is linear in the size of the circuit [De Raedt and Kimmig, 2015; Fierens *et al.*, 2015]. Furthermore, the circuits only need to

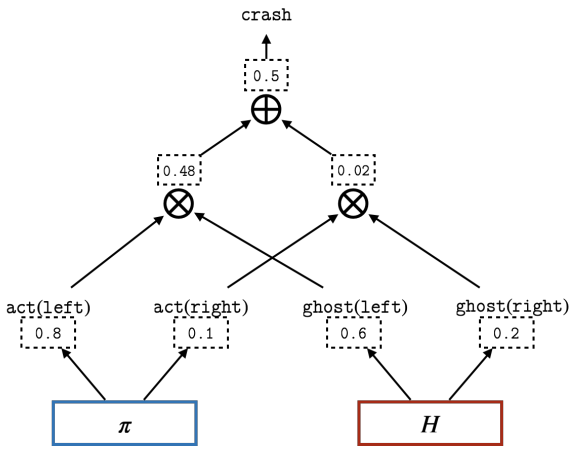

Figure 2: A circuit compiled from a program.

be compiled once and then can be reused to compute gradients for all states. As an example, the following program is compiled into the circuit in Fig. 2.

```
0.2::act(dn);
0.6::act(left);
0.2::act(right).
0.8::ghost(left).  0.1::ghost(right).
crash:- act(left), ghost(left).
crash:- act(right), ghost(right).
```

Although we are using a probabilistic logic programming language (i.e. ProbLog), we could also have used alternative representations such as Bayesian networks, or other StarAI models [De Raedt *et al.*, 2016] to reason about safety. Currently, ProbLog is limited to discrete actions, but it should be possible to model continuous actions using extensions in future work [Nitti *et al.*, 2016].

5 Probabilistic Logic Policy Gradient

This section demonstrates how to use probabilistic logic shields with a deep reinforcement learning method. We will focus on real-vector states (such as images), while specifying the safety constraint symbolically and logically. Our goal is to maximize a linear combination of $\mathbf{P}_\pi(\texttt{safe}|s)$ and the total expected return along the trajectory. We propose a novel safe policy gradient technique, named *Probabilistic Logic Policy Gradient (PLPG)*. PLPG integrates probabilistic logic shields with policy gradients and guarantees convergence to a safe and optimal policy if one exists.

5.1 PLPG for Probabilistic Shielding

To integrate probabilistic logic shields with policy gradient, we replace the base policy in Eq. (1) with the shielded policy obtained in Eq. (3). This gradient is called the *shielded policy gradient*.

$$\mathbb{E}_{\pi_\theta^+}[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta^+(a_t|s_t)] \tag{4}$$

Replacing π with π^+ requires the shield to be differentiable, meaning that the derivative of the policy $\nabla_\theta \log \pi_\theta^+(a_t|s_t)$ with respect to θ exists at every point. We will later see that rejection-based shields in Eq. (7) are not differentiable and cannot be applied in Eq. (4). The shielded policy gradient encourages policies that yield higher rewards, indicated by a larger Ψ, in the same manner as a standard policy gradient. It assumes that unsafe actions have been filtered by the shield. However, when the safety specification is uncertain, unsafe actions may still be taken, and the gradient may still encourage unsafe actions that were not filtered by the shield and yielded a high return.

To address this issue, we introduce a safety loss to penalize unsafe policies. The safety loss is designed such that a safe policy will have a small loss and a completely safe policy will have a loss of zero. The loss is expressed by interpreting the policy safety as *the probability that π^+ satisfies the safety constraint*, i.e. $-\log \mathbf{P}_{\pi^+}(\texttt{safe}|s)$, using the semantic loss approach in [Xu *et al.*, 2018]. We define the corresponding *safety gradient* as:

$$-\mathbb{E}_{\pi_\theta^+}[\nabla_\theta \log \mathbf{P}_{\pi^+}(\texttt{safe}|s)] \tag{5}$$

Notice that the safety gradient represents a regret and is not a shielding mechanism. By combining the shielded policy gradient and the safety gradient, we introduce a new Safe RL technique.

Definition 5.1. ***(PLPG)*** *The probabilistic logic policy gradient $\nabla_\theta J(\theta)$ is*

$$\mathbb{E}_{\pi_\theta^+}[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta^+(a_t|s_t) - \alpha \nabla_\theta \log \mathbf{P}_{\pi_\theta^+}(\texttt{safe}|s_t)] \tag{6}$$

where $\alpha \in [0, \infty)$ is a hyperparameter that controls the combination of the two gradients.

The hyperparameter α guides the search towards an optimal and safe policy. Both gradients in PLPG are essential. The shielded policy gradient ensures that the agent avoids immediate danger, while the safety gradient penalizes unsafe behavior. The interaction between shielded and loss-based gradients is still a relative new topic, but recent advancements in neuro-symbolic learning have highlighted the importance of considering both [Ahmed *et al.*, 2022]. Our experiments will show that relying on one but not the other is practically insufficient.

5.2 Probabilistic vs Rejection-based Shielding

Finding a policy in the safe policy space cannot be solved by rejection-based shielding (cf. Fig. 3, left) without assuming that a state-action pair is either completely safe or unsafe. In fact, it is often assumed that a set of safe state-action pairs $SA_{\texttt{safe}}$ is given by the user. Implementing a rejection-based shield requires repeatedly sampling from a base policy until an action is accepted. However, this approach relies on inefficient rejection sampling schemes, which lack a clear link to probabilistic semantics [Robert *et al.*, 1999].

$$\pi^+(a|s) = \frac{\mathbb{1}_{[(s,a)\in SA_{\texttt{safe}}]}\pi(a|s)}{\sum_{a'\in A} \mathbb{1}_{[(s,a')\in SA_{\texttt{safe}}]}\pi(a'|s)} \tag{7}$$

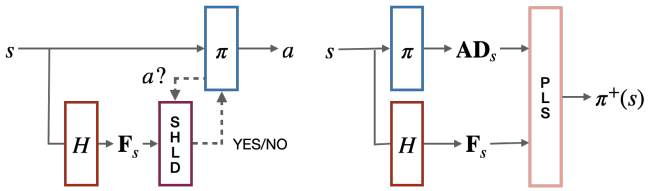

Figure 3: A comparison between a traditional rejection-based shield SHLD (left) and a Probabilistic Logic Shield PLS (right). Both shields interact with the same policy π and noisy sensors H to compute a safe action. **Left**: π must keep sampling actions until a safe action is accepted by SHLD. This requires an assumption that an action is either completely safe or unsafe. **Right**: We replace SHLD with PLS that proposes a safer policy π^+ on the policy level without imposing the assumption.

Convergence Under Perfect Safety Information

It is common for existing shielding approaches to combine policy gradients with a rejection-based shield (i.e. Eq. (7)), leading to the following form of policy gradient.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta^+}[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(s_t, a_t)] \tag{8}$$

This approach has a known issue of potentially learning sub-optimal policies, even when given perfect safety information [Kalweit *et al.*, 2020; Ray *et al.*, 2019; Hunt *et al.*, 2021; Anderson *et al.*, 2020]. The issue arises from a policy mismatch between π^+ and π in Eq. (8). Specifically, Eq. (8) is an *off-policy* algorithm, where the policy used to explore (i.e. π^+) differs from the policy being updated (i.e. π)[1]. For any off-policy policy gradient method to converge to an optimal policy (even in the tabular case), the behavior policy (used for exploration) and the target policy (to be updated) must appropriately visit the same state-action space, i.e., if $\pi(a|s) > 0$ then $\pi^+(a|s) > 0$ [Sutton and Barto, 2018]. However, this requirement is violated by Eq. (8).

In contrast, PLPG simplifies to Eq. (4) when given perfect sensor information. Since Eq. (4) shares the same form as the standard policy gradient (Eq. (1)), PLPG is guaranteed to converge to an optimal policy according to the Policy Gradient Theorem [Sutton *et al.*, 2000].

Proposition 2. *PLPG, i.e. Eq.* (6)*, converges to an optimal policy given perfect safety information in all states.*

It is important to note that the base policy π_θ learned by PLPG (cf. Eq. (6)) is generally not equivalent to the one learned with a rejection-based shield (cf. Eq. (8)), assuming all other factors remain equal. In Eq. (8), the parameters θ depend on the base policy. However, in Eq. (6), the parameters θ depend on the shielded policy π^+, and the base policy is learned in a way that optimizes π^+ *given* the safety constraints imposed by the shield.

Learning From Unsafe Actions

PLPG has a significant advantage over a rejection-based shield as it learns from not only the actions accepted by the shield but *all available actions* in the current state This means that safety information about the rejected actions can be incorporated without having to execute them. By conditioning the action probabilities on the `safety` atom (cf. Eq. (3)), the probabilistic logic shield implicitly performs planning and exploits this safety information to update the policy through backpropagation.

[1]A well-known off-policy technique is Q-learning where the behavior policy different from the target policy.

6 Experiments

We conduct an empirical evaluation of PLPG, comparing it to other baselines in terms of return and safety in a reinforcement learning setting. The source code can be found on https://github.com/wenchiyang/pls.

All experiments are run for a number of episodes until the agent has taken a total of 600k actions. All episodes starts with the same initial states and ends when the agent reaches a goal state or an unsuccessful absorbing state, or at the maximum length of an episode. We will measure the return and and whether the constraint is violated for all episodes. All agents in all domains are trained using 600k learning steps and each experiment is repeated five times using different seeds.

Experimental Setup Experiments are run in three environments. (1) *Stars*, where the agent must collect as many stars as possible without going into a stationary fire; (2) *Pacman*, where the agent must collect stars without getting caught by the fire ghosts moving around, and (3) *Car Racing*, where the agent must follow the track without driving into the grass area. For each domain, we use two configurations. A detailed description of these configurations can be found in Appendix A.3.

We compare PLPG to three RL baselines.

- PPO: a standard safety-agnostic agent that starts with a random initial policy [Schulman *et al.*, 2017].
- VSRL: an agent augmented with a deterministic rejection-based shield [Hunt *et al.*, 2021]. Its structure is shown in Fig. 3 (left).
- ϵ-VSRL: we introduce a risk-taking variant of VSRL that has a small ϵ probability of accepting any action, akin to ϵ-greedy [Fernández and Veloso, 2006]. By simulating artificial sensor noises, ϵ-VSRL is expected to enhance the ability of VSRL to handle noisy environments.

To ensure a fair comparison, we equip all safety-aware agents, including VSRL, ϵ-VSRL and PLPG, with identical safety sensors. PPO agents do not have any safety sensors. It should be noted that PLPG does not assume a complete model of the environment. Hence, we do not compare to shielding approaches that rely on such models. In our setting, the environment model is unknown. Instead, we assume a safety model, which is an abstraction of the underlying model (cf. Section 4). Further investigation is required to understand the implications of the relation between of the safety model and the underlying MDP.

We aim to answer the following questions:

Q1 *Does PLPG outperform its competitors in terms of safety and return?*

Q2 *What is the effect of the hyperparameters α and ϵ?*

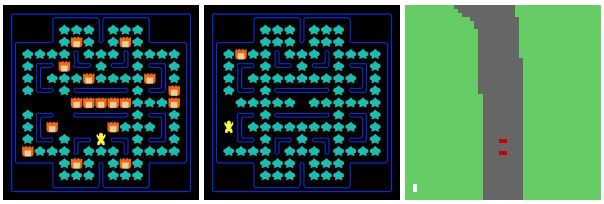

Figure 4: The domains of Stars, Pacman and Car Racing. The fires in Stars remain stationary and the ones in Pacman move around.

Q3 *Does the shielded policy gradient or the safety gradient have a stronger influence on safety in* PLPG*?*

Q4 *What is the computational cost of a multi-step safety look-ahead in* PLPG*?*

Metrics All agents are compared based on two metrics: (1) *average normalized return*, i.e. the per-episode return averaged over the last 100 episodes and then normalized to $[0, 1]$; and (2) *cumulative normalized violation*, i.e. the total number of constraint violations normalized to $[0, 1]$. Detailed numerical results can be found in Appendix B.

Probabilistic Safety via Noisy Sensors We consider a set of noisy sensors around the agent that provide local safety information. For instance, the four fire sensor readings in Fig. 4 (left) might be $\{0.6 :: \texttt{fire}(0, 1).\ 0.1 :: \texttt{fire}(0, -1).\ 0.1 :: \texttt{fire}(-1, 0).\ 0.4 :: \texttt{fire}(1, 0).\}$ The PLPG agents can directly use the noisy sensor readings. However, VSRL and ϵ-VSRL agents require deterministic sensor readings. Hence, the readings must be discretized to $\{0, 1\}$, resulting in $\{1 :: \texttt{fire}(0, 1).\ 0 :: \texttt{fire}(0, -1).\ 0 :: \texttt{fire}(-1, 0).\ 0 :: \texttt{fire}(1, 0).\}$ In all domains, the noisy sensors are standard pre-trained neural approximators of an accuracy exceeding 99%. The detailed training procedure can be found in Appendix B.1. Despite the high accuracy of these sensors, we will show that discretizing sensor readings is harmful in terms of safety.

Q1: Lower Violation and Higher Return We evaluate the safety and return of PLPG compared to the baselines by measuring the total safety violations and the average episodic returns. However, before doing so, we must select appropriate hyperparameters for PLPG and ϵ-VSRL agents. Since our goal is to find an optimal policy in the safe policy space, we select values for α and ϵ that result in the lowest violation in each domain. We consider $\alpha \in \{0, 0.1, 0.5, 1, 5\}$ and $\epsilon \in \{0, 0.005, 0.01, 0.05, 0.1, 0.2, 0.5, 1.0\}$. These hyperparameter choices are can be found in Appendix B, and they will remain fixed for the rest of the experiments, except for Q2 where we explicitly analyze their effects.

The results are visualized in Fig. 5, where each data point corresponds to a policy trained for 600k steps. For analysis purposes, we will report the average return and constraint violations of all six configurations across all domains.

Our results show that the PLPG agents achieve the lowest violation while maintaining a comparable return to the other agents. When augmented with perfect sensors, the violation of PLPG is 50.2% lower than PPO and 25.7% lower than VSRL. When augmented with noisy sensors, the violation of PLPG is 51.3% lower than PPO, 24.5% lower than

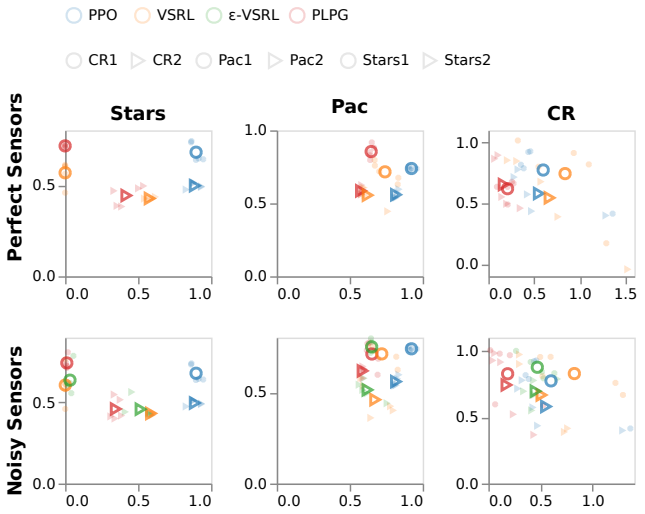

Figure 5: Trade-off between **Violation (x-axis) and Return (y-axis)**. Each small data point is an agent's policy and the large data points are the average of five seeds. Each domain has two configurations Table 2. An ideal policy should lie in the upper-left corner.

VSRL and 13.5% lower than ϵ-VSRL. Our results also illustrates that PLPG achieves a comparable (or slightly higher) return while having the lowest violation. When augmented with perfect sensors, the return of PLPG is 0.5% higher than PPO and 4.8% higher than VSRL. When augmented with noisy sensors, the return of PLPG is 4.5% higher than PPO, 6.5% higher than VSRL and 6.7% higher than ϵ-VSRL.

Compared to the other domains, the Car Racing domain has more complex and continuous actions effects, and the safety sensors do not capture the inertia of the car, which involves the velocity and the underlying physical mechanism such as friction. Therefore, in CR, relying solely on safety sensor readings is insufficient for the car to avoid driving into the grass. In domains where safety sensors are insufficient, the agent must be able to take slightly unsafe actions to learn. This is possible for agents like PPO, ϵ-VSRL and PLPG, whereas VSRL solely relies on sensors. Hence, comparing to the safety-agnostic baseline PPO, both ϵ-VSRL and PLPG significantly reduce violations by 37.8% and 50.2%. In contrast, VSRL leads to 18% more violations even when given perfect safety sensors. Note that we measure the actual violations instead of policy safety $P_\pi(\texttt{safe}|s)$. The policy safety during the learning process is plotted in Appendix B.

Q2: Selection of Hyperparameters α and ϵ We analyze the hyperparameters α and ϵ to evaluate their impact on the performance of PLPG and ϵ-VSRL in noisy environments. We measure the episodic return and constraint violations. The normalized results are plotted in Fig. 6 and the detailed numbers can be found in Appendix B.

The effect of different values of α on PLPG agents is evident. Increasing α leads to a convex trend in both the return and the violation counts. For all the domains, the optimal value of α generally falls between 0.1 and 1, where the constraint violations are minimized and the return is optimal. This illustrates the benefits of combining the shielded pol-

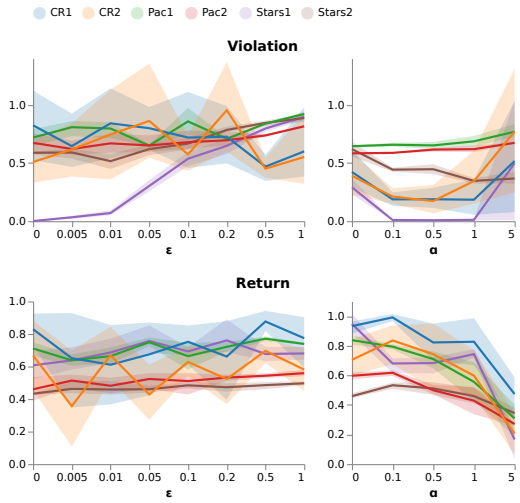

Figure 6: The episodic return and cumulative constraint violation of VSRL (left) and PLPG (right) agents in noisy environments. **Left**: The effect of ϵ is insignificant. **Right**: Increasing α leads to a convex trend in both return and violation.

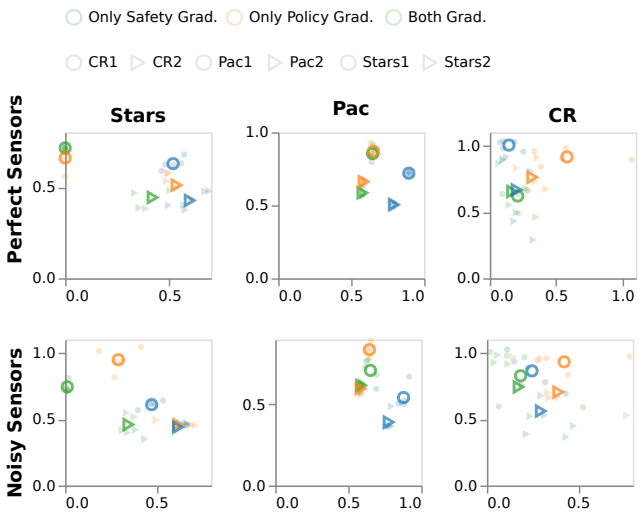

Figure 7: Trade-off between **Violation (x-axis) and Return (y-axis)**. Each data point is an agent's policy and the big data points are the average of five seeds. Each domain has two configurations Table 2. An ideal policy should lie in the upper-left corner.

icy gradient and the safety gradient in PLPG. In contrast, the effect of ϵ on VSRL agents is insignificant. Increasing ϵ generally improves return but worsens constraint violations. This indicates that simply randomizing the policy is not effective in improving the agent's ability to handle noisy environments. Notice that there is no one-to-one mapping between the two hyperparameters, as α controls the combination of the gradients and ϵ controls the degree of unsafe exploration. Overall, this analysis emphasizes the benefits of directly utilizing noisy sensor readings as opposed to relying on random exploration.

Q3: PLPG Gradient Analysis We evaluate the interaction between the shielded policy gradient and the safety gradient by introducing two ablated agents: one that uses only safety gradients (cf. Eq. (4)) and the other that uses only shielded policy gradients (cf. Eq. (5)). The results are plotted in Fig. 7 where each data point corresponds to a policy trained for 600k steps. For analysis purposes, we will report the average return and constraint violations of all six configurations across all domains.

Our results show that combining both gradients results in a safer learning process compared to using only one gradient. When augmented with perfect sensors, using both gradients leads to 24.7% fewer violations than using only safety gradients, and 10.8% fewer violations than using only shielded policy gradients. When augmented with noisy sensors, using both gradients leads to 25.5% fewer violations than using only safety gradients, and 15.5% fewer violations than using only shielded policy. However, the relative importance of the two gradients varies across domains. In Stars and Pacman, using only policy gradients causes leads to fewer violations compared to using only safety gradients. Conversely, in CR, using only safety gradients leads to fewer violations, which is a consequence of the inertia effect, as discussed in Q1.

Safety horizon	1 step	2 steps	3 steps	4 steps
#sensors	4	12	24	40
Circuit size	116	1073	8246	373676
Compilation (s)	0.29	0.70	1.70	15.94
Evaluation (s)	0.01	0.08	0.66	27.39
Return/Violation	0.81 / 0.82	0.85 / 0.65	**0.86 / 0.57**	-

Table 1: Multi-step Safety Look-ahead

Q4: Multi-step Safety Look-ahead We analyze the behaviour of PLPG when employing probabilistic logic shields for multi-step safety look-ahead. This approach requires additional sensors around the agent to detect potential dangers over a larger safety horizon. For example, in Pacman, new sensors are required to detect the presence of a ghost at a distance of N units where N is the safety horizon. We evaluate the performance of PLPG in terms of return, violation and computational cost. The results are shown in Table 1. As the horizon increases, there is an exponential growth in size of the compiled circuit and the corresponding inference time. However, there is a significant improvement in both safety and return, especially when transitioning from one-step to a two-step horizon.

7 Related Work

Safe RL. Safe RL aims to avoid unsafe consequences through the use of various safety representations [García and Fernández, 2015]. There are several ways to achieve this goal. One could constrain the expected cost [Achiam *et al.*, 2017; Moldovan and Abbeel, 2012], maximize safety constraint satisfiability through a loss function [Xu *et al.*, 2018], add a penalty to the agent when the constraint is violated [Pham *et al.*, 2018; Tessler *et al.*, 2019; Memarian

et al., 2021], or construct a more complex reward structure using temporal logic [De Giacomo *et al.*, 2021; Camacho *et al.*, 2019; Jiang *et al.*, 2021; Hasanbeig *et al.*, 2019; Den Hengst *et al.*, 2022]. These approaches express safety as a loss, while our method directly prevents the agent from taking actions that can potentially lead to safety violation.

Probabilistic Shields. Shielding is a safe reinforcement learning approach that aims to completely avoid unsafe actions during the learning process [Alshiekh *et al.*, 2018; Jansen *et al.*, 2020]. Previous shielding approaches have been limited to symbolic state spaces and are not suitable for noisy environments [Jansen *et al.*, 2020; Harris and Schaub, 2020; Hunt *et al.*, 2021; Anderson *et al.*, 2020]. To address uncertainty, some methods incorporate randomization, e.g. simulating future states in an emulator to estimate risk [Li and Bastani, 2020; Giacobbe *et al.*, 2021], using ϵ-greedy exploration that permits unsafe actions [García and Fernández, 2019], or randomizing the policy based on the current belief state [Karkus *et al.*, 2017]. To integrate shielding with neural policies, one can translate a neural policy to a symbolic one that can be formally verified [Bastani *et al.*, 2018; Verma *et al.*, 2019]. However, these methods rely on sampling and do not have a clear connection to uncertainty present in the environment while our method directly exploits such uncertainty through the use of probabilistic logic programming principles. Belief states can capture uncertainty but require an environment model to keep track of the agent's belief state, which is a stronger assumption than our method [Junges *et al.*, 2021; Carr *et al.*, 2022].

Differentiable layers for neural policies. The use of differentiable shields has gain some attention in the field. One popular approach is to add a differentiable layer to the policy network to prevent constraint violations. Most of these methods focus on smooth physical rules [Dalal *et al.*, 2018; Pham *et al.*, 2018; Cheng *et al.*, 2019] and only a few involve logical constraints. In [Kimura *et al.*, 2021], an optimization layer is used to transform a state-value function into a policy encoded as a logical neural network. [Ahmed *et al.*, 2022] encode differentiable and hard logical constraints for neural networks using probabilistic logic programming. While being similar, this last model focuses on prediction tasks and do not consider a trade-off between return and constraint satisfiability.

8 Conclusion

We have introduced Probabilistic Logic Policy Gradient, a novel class of end-to-end differentiable shielding techniques. Probabilistic Logic Policy Gradient enables efficient training of safe-by-construction neural policies by incorporating probabilistic logic shields on top of the standard neural policy. It is a generalization of classical shielding [Hunt *et al.*, 2021] that allows for both deterministic and probabilistic safety specifications. Future work will be dedicated to extending probabilistic logic shields to be compatible with a larger class of RL algorithms, including those that employ continuous policies.

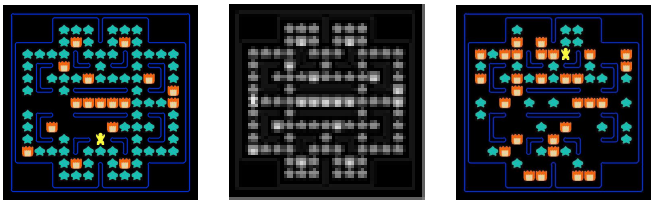

Figure 8: **Left**: A Stars image. **Middle**: A Stars state (downsampled from the left). **Right**: A example to pre-train noisy sensors.

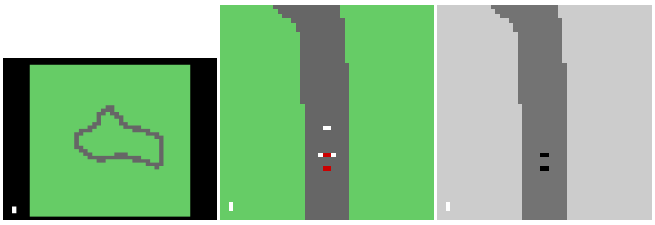

Figure 9: **Left**: A lap in Car Racing. **Middle**: A Car Racing image. We mark the sensors around the agent white but they not visible in training. **Right**: A Car Racing state (downsampled from the middle).

A Environments

All environments in this work are available on GitHub under the MIT license. The source code and the data used to generate Figs. 5 to 7 can be found on https://github.com/wenchiyang/pls.

A.1 Stars and Pacman

We build the Stars environments using Berkeley's Pac-Man Project [2]. Our environment is a 15 by 15 grid world containing stars and fires, e.g. Fig. 8, where the agent can move around using five discrete, deterministic actions: stay, up, down, left, right. Each action costs a negative reward of -0.1. The task of the agent is to collect as many stars as possible in an episode. Collecting one star yields a reward of 1. When the agent finds all stars, the episode ends with a reward of 10. If the agent crashes into fire or has taken 200 steps (maximum episode length), the episode ends with no rewards. The same map is reused for all episodes. Each state is a downsampled, grayscale image where each pixel is a value between -1 and 1, e.g. Fig. 8 (right).

Pacman is more complicated than the Stars in that the fire ghosts can move around and their transition models are unknown. All the other environmental parameters are the same as Stars.

A.2 Car Racing

We simplify Gym's Car Racing environment [Brockman *et al.*, 2016]. Our environment is a randomly generated car track, e.g. Fig. 9 (left). At the beginning, the agent will be put on the track, e.g. Fig. 9 (middle). The task of the agent is to drive around the track (i.e. to finish one lap) within 1000 frames. In each frame, the agent takes a discrete action: do-nothing, accelerate, brake, turn-left or turn-right. Each ac-

[2] http://ai.berkeley.edu/project_overview.html

	Description	Return Range	Violation Range	Perfect Sensors	Noisy Sensors	
				α	ϵ	α
Stars1	Deterministic actions	[0, 45]	[0, 15k]	0.5	0.005	1
Stars2	Stochastic actions			1	0.01	1
Pacman1	One ghost	[0, 40]	[0, 7k]	0.1	0.05	0.5
Pacman2	Two ghosts		[0, 15k]	0.5	0.005	0.1
Car Racing1	Smoother track	[0, 900]	[0, 1]	0.5	0.5	1
Car Racing2	Curlier track			0.1	0.5	0.5

Table 2: Summary of configurations.

tion costs a negative reward of -0.1. The agent gets a small reward if it continues to follow the track. The agent does not get a penalty for driving on the grass, however, if the car drives outside of the map (i.e. the black part in Fig. 9, left), the episode ends with a negative reward of -100. The same map is reused for all episodes. Each state is a downsampled, grayscale image where each pixel is a value between -1 and 1, e.g. Fig. 9 (right).

The original environment has a continuous action space that consists of three parameters: steering, gas and break with the minimum values $[-1, 0, 0]$ and the maximum values $[+1, +1, +1]$. In this paper, all agents use five discrete actions: do-nothing ($[0, 0, 0]$), accelerate ($[0, 1, 0]$), brake ($[0, 0, 0.8]$), turn-left ($[-1, 0, 0]$), turn-right ($[1, 0, 0]$).

We use the following probabilistic logic shield. The `act/1` predicates represent the base policy and the `grass/1` predicates represent whether there is grass in front, on the left or right hand side of the agent. In Fig. 9 (middle), the sensors may produce $\{$`0 :: grass(0)`, `0 :: grass(1)` `0 :: grass(2)`$\}$.

A.3 Configurations of Domains

We run two configurations for each domain. The second configuration is more challenging than the first one. Stars1 is a deterministic environment. Stars2 is a stochastic environment where the agent has a small probability to enter an unintended neighboring cell. For example, after performing `up`, the agent will enter the intended cell $(0, 1)$ with a probability of 0.9 and it will enter $(-1, 0)$ and $(1, 0)$ with a probability of 0.05. Pacman1 has one moving fire ghost and Pacman2 has two. Car Racing1 has a smother track and Car Racing2 has a curlier track, which is more difficult to learn.

All configurations may have different normalization ranges and hyperparameters. We select the hyperparameters that result in the lowest violations in each configuration. Table 2 lists a summary of all configurations and all other tables follows this table if not explicitly specified.

B Extra Experimental Information

Experiments are run on machines that consist of Intel(R) Xeon(R) E3-1225 CPU cores and 32Gb memory. All agents are trained using PPO in `stable-baselines` [Hill *et al.*, 2018] with batch size=512, n_epochs=15, n_steps=2048, clip range=0.1, learning rate=0.0001. All policy networks and value networks have two hidden layers of size 64. All the other hyperparameters are set to default as in `stable-baselines` [Hill *et al.*, 2018]. We train all agents in all configurations using 600k learning steps and all experiments are repeated using five different seeds.

B.1 Approximating Noisy Sensors

We approximate noisy sensors using convolutional neural networks. In all the environments, we use 4 convolutional layers with respectively 8, 16, 32, 64 (5x5) filters and relu activations. The output is computed by a dense layer with sigmoid activation function. The number of output neurons depends on the ProbLog program for that experiment (see Appendix A.1). We pre-train the networks and then we fix them during the reinforcement learning process. The pre-training strategy is the following. We generated randomly 3k images for Stars and Pacman with 30 fires and 30 stars, e.g. Fig. 8 (right), and 2k images for Car Racing, e.g. Fig. 9 (right). We selected the size of the pre-training datasets in order to achieve an accuracy higher than 99% on a validation set of 100 examples.

Acknowledgements

This work was supported by the Research Foundation - Flanders under EOS No. 309925744, the Flemish Government (AI Research Program), the EU Horizon 2020 programme TAILOR under No. 952215, and the KU Leuven Research fund. GM has received funding from FWO (1239422N). LDR is partially funded by the Wallenberg AI, Autonomous Systems and Software Program.

References

[Achiam *et al.*, 2017] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 22–31. JMLR.org, 2017.

[Ahmed *et al.*, 2022] Kareem Ahmed, Stefano Teso, Kai-Wei Chang, Guy Van den Broeck, and Antonio Vergari. Semantic probabilistic layers for neuro-symbolic learning. In *Advances in Neural Information Processing Systems*, 2022.

[Alshiekh *et al.*, 2018] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott

Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018.

[Anderson *et al.*, 2020] Greg Anderson, Abhinav Verma, Isil Dillig, and Swarat Chaudhuri. Neurosymbolic reinforcement learning with formally verified exploration. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, 2020.

[Bastani *et al.*, 2018] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 2499–2509. Curran Associates Inc., 2018.

[Brockman *et al.*, 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[Camacho *et al.*, 2019] Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, 7 2019.

[Carr *et al.*, 2022] Steven Carr, Nils Jansen, Sebastian Junges, and Ufuk Topcu. Safe reinforcement learning via shielding for pomdps. *CoRR*, abs/2204.00755, 2022.

[Cheng *et al.*, 2019] Richard Cheng, Gábor Orosz, Richard M. Murray, and Joel W. Burdick. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019.

[Dalal *et al.*, 2018] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerík, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *CoRR*, abs/1801.08757, 2018.

[Darwiche, 2003] Adnan Darwiche. A differential approach to inference in bayesian networks. *J. ACM*, 50(3):280–305, may 2003.

[De Giacomo *et al.*, 2021] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications. *Proceedings of the International Conference on Automated Planning and Scheduling*, 29(1):128–136, 2021.

[De Raedt and Kimmig, 2015] Luc De Raedt and Angelika Kimmig. Probabilistic (logic) programming concepts. *Mach. Learn.*, 100(1):5–47, 2015.

[De Raedt *et al.*, 2016] Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. *Statistical Relational Artificial Intelligence: Logic, Probability, and Computation*, volume 32 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool, 2016.

[Den Hengst *et al.*, 2022] Floris Den Hengst, Vincent François-Lavet, Mark Hoogendoorn, and Frank van Harmelen. Planning for potential: Efficient safe reinforcement learning. *Mach. Learn.*, 111(6):2255–2274, 2022.

[Fernández and Veloso, 2006] Fernando Fernández and Manuela Veloso. Probabilistic policy reuse in a reinforcement learning agent. AAMAS '06. Association for Computing Machinery, 2006.

[Fierens *et al.*, 2015] Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming*, 15:358–401, 5 2015.

[García and Fernández, 2015] Javier García and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16:1437–1480, 2015.

[García and Fernández, 2019] Javier García and Fernando Fernández. Probabilistic policy reuse for safe reinforcement learning. *ACM Trans. Auton. Adapt. Syst.*, 13(3), 2019.

[Giacobbe *et al.*, 2021] Mirco Giacobbe, Mohammadhosein Hasanbeig, Daniel Kroening, and Hjalmar Wijk. Shielding atari games with bounded prescience. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, 2021.

[Harris and Schaub, 2020] Andrew Harris and Hanspeter Schaub. Spacecraft command and control with safety guarantees using shielded deep reinforcement learning. *AIAA Scitech 2020 Forum*, 1 PartF, 2020.

[Hasanbeig *et al.*, 2019] M. Hasanbeig, Y. Kantaros, A. Abate, D. Kroening, G. J. Pappas, and I. Lee. Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019.

[Hill *et al.*, 2018] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018. Accessed: 2023-01-18.

[Hunt *et al.*, 2021] Nathan Hunt, Nathan Fulton, Sara Magliacane, Trong Nghia Hoang, Subhro Das, and Armando Solar-Lezama. Verifiably safe exploration for end-to-end reinforcement learning. In *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, HSCC '21, 2021.

[Jansen *et al.*, 2020] Nils Jansen, Bettina Könighofer, Sebastian Junges, Alex Serban, and Roderick Bloem. Safe reinforcement learning using probabilistic shields. In *31st International Conference on Concurrency Theory, CONCUR 2020*, 2020.

[Jiang *et al.*, 2021] Yuqian Jiang, Suda Bharadwaj, Bo Wu, Rishi Shah, Ufuk Topcu, and Peter Stone. Temporal-logic-

based reward shaping for continuing reinforcement learning tasks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(9):7995–8003, 2021.

[Junges *et al.*, 2021] Sebastian Junges, Nils Jansen, and Sanjit A. Seshia. Enforcing almost-sure reachability in pomdps. In *Computer Aided Verification*. Springer International Publishing, 2021.

[Kalweit *et al.*, 2020] Gabriel Kalweit, Maria Huegle, Moritz Werling, and Joschka Boedecker. Deep inverse q-learning with constraints. In *Advances in Neural Information Processing Systems*, volume 33, pages 14291–14302. Curran Associates, Inc., 2020.

[Karkus *et al.*, 2017] Peter Karkus, David Hsu, and Wee Sun Lee. Qmdp-net: Deep learning for planning under partial observability. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17. Curran Associates Inc., 2017.

[Kimura *et al.*, 2021] Daiki Kimura, Subhajit Chaudhury, Akifumi Wachi, Ryosuke Kohita, Asim Munawar, Michiaki Tatsubori, and Alexander Gray. Reinforcement Learning with External Knowledge by using Logical Neural Networks. 2021.

[Li and Bastani, 2020] Shuo Li and Osbert Bastani. Robust Model Predictive Shielding for Safe Reinforcement Learning with Stochastic Dynamics. *Proceedings - IEEE International Conference on Robotics and Automation*, 2020.

[Memarian *et al.*, 2021] Farzan Memarian, Wonjoon Goo, Rudolf Lioutikov, Scott Niekum, and Ufuk Topcu. Self-supervised online reward shaping in sparse-reward environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 2369–2375. IEEE, 2021.

[Mnih *et al.*, 2016] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1928–1937. JMLR.org, 2016.

[Moldovan and Abbeel, 2012] Teodor Mihai Moldovan and Pieter Abbeel. Safe exploration in markov decision processes. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12. Omnipress, 2012.

[Nitti *et al.*, 2016] Davide Nitti, Tinne De Laet, and Luc De Raedt. Probabilistic logic programming for hybrid relational domains. *Machine Learning*, 103:407–449, 2016.

[Pham *et al.*, 2018] Tu-Hoa Pham, Giovanni De Magistris, and Ryuki Tachibana. Optlayer - practical constrained optimization for deep reinforcement learning in the real world. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6236–6243, 2018.

[Pishro-Nik, 2014] H. Pishro-Nik. *Introduction to Probability, Statistics, and Random Processes*. Kappa Research, LLC, 2014.

[Ray *et al.*, 2019] Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking Safe Exploration in Deep Reinforcement Learning. *arXiv preprint*, pages S. 1–6, 2019.

[Robert *et al.*, 1999] Christian P Robert, George Casella, and George Casella. *Monte Carlo statistical methods*, volume 2. Springer, 1999.

[Schulman *et al.*, 2015] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2015.

[Schulman *et al.*, 2016] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *4th International Conference on Learning Representations, ICLR*, 2016.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[Sutton and Barto, 2018] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, 2018.

[Sutton *et al.*, 2000] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 2000.

[Tessler *et al.*, 2019] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. Reward constrained policy optimization. In *7th International Conference on Learning Representations, ICLR*. OpenReview.net, 2019.

[Verma *et al.*, 2019] Abhinav Verma, Hoang M. Le, Yisong Yue, and Swarat Chaudhuri. Imitation-projected programmatic reinforcement learning. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019.

[Xu *et al.*, 2018] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *Proceedings of the 35th International Conference on Machine Learning, ICML'18*. PMLR, 2018.