Neurosymbolic Reinforcement Learning with Formally Verified Exploration

Greg Anderson
UT Austin
ganderso@cs.utexas.edu

Abhinav Verma
UT Austin
verma@utexas.edu

Isil Dillig
UT Austin
isil@cs.utexas.edu

Swarat Chaudhuri
UT Austin
swarat@cs.utexas.edu

Abstract

We present REVEL, a partially neural reinforcement learning (RL) framework for provably safe exploration in continuous state and action spaces. A key challenge for provably safe deep RL is that repeatedly verifying neural networks within a learning loop is computationally infeasible. We address this challenge using two policy classes: a general, neurosymbolic class with approximate gradients and a more restricted class of symbolic policies that allows efficient verification. Our learning algorithm is a mirror descent over policies: in each iteration, it safely lifts a symbolic policy into the neurosymbolic space, performs safe gradient updates to the resulting policy, and projects the updated policy into the safe symbolic subset, all without requiring explicit verification of neural networks. Our empirical results show that REVEL enforces safe exploration in many scenarios in which Constrained Policy Optimization does not, and that it can discover policies that outperform those learned through prior approaches to verified exploration.

1 Introduction

Guaranteeing that an agent behaves safely during exploration is a fundamental problem in reinforcement learning (RL) [13, 1]. Most approaches to the problem are based on stochastic definitions of safety [23, 7, 1, 7], requiring the agent to satisfy a safety constraint with high probability or in expectation. However, in applications such as autonomous robotics, unsafe agent actions — no matter how improbable — can lead to cascading failures with high human and financial costs. As a result, it can be important to ensure that the agent behaves safely *even on worst-case inputs*.

A number of recent efforts [3, 11] use formal methods to offer such worst-case guarantees during exploration. Broadly, these methods construct a space of provably safe policies *before* the learning process starts. Then, during exploration, a *safety monitor* observes the learner, forbidding all actions that cannot result from one of these safe policies. If the learner is about to take a forbidden action, a safe policy (a *safety shield*) is executed instead.

So far, these methods have only been used to discover policies over simple, finite action spaces. Using them in more complex settings — in particular, continuous action spaces — is much more challenging. A key problem is that these safety monitors are constructed a priori and are blind to the internal state of the learner. As we experimentally show later in this paper, such a "one-size-fits-all" strategy can unnecessarily limit exploration and impede learner performance.

In this paper, we improve this state of the art through an RL framework, called REVEL[1] that allows learning over continuous state and action spaces, supports (partially) neural policy representations and contemporary policy gradient methods for learning, while also ensuring that every intermediate policy that the learner constructs during exploration is safe on worst-case inputs. Like previous efforts, REVEL uses monitoring and shielding. However, unlike in prior work, the monitor and the shield are updated as learning progresses.

A key feature of our approach is that we repeatedly invoke a formal verifier from within the learning loop. Doing this is challenging because of the high computational cost of verifying neural networks. We overcome this challenge using a neurosymbolic policy representation in which the shield and the monitor are expressed in an easily-verifiable symbolic form, whereas the normal-mode policy is given by a neural network. Overall, this representation admits efficient gradient-based learning as well as efficient updates to both the shield and monitor.

To learn such neurosymbolic policies, we build on PROPEL [29], a recent RL framework in which policies are represented in compact symbolic forms (albeit without consideration of safety), and design a learning algorithm that performs a functional mirror descent in the space of neurosymbolic policies. The algorithm views the set of shields as being obtained by imposing a constraint on the general policy space. Starting with a safe but suboptimal shield, it alternates between: (i) safely *lifting* the current shield into the unconstrained policy space by adding a neural component; (ii) safely *updating* this neurosymbolic policy using approximate gradients; and (iii) using a form of imitation learning to *project* the updated policy back into the constrained space of shields. Importantly, none of these steps requires direct verification of neural networks.

Our empirical evaluation, on a suite of continuous control problems, shows that REVEL enforces safe exploration in many scenarios where established RL algorithms (including CPO [1], which is motivated by safe RL) do not, while discovering policies that outperform policies based on static shields. Also, building on results for PROPEL, we develop a theoretical analysis of REVEL.

In summary, the contributions of the paper are threefold. First, we introduce the first RL approach to use deep policy representations and policy gradient methods while guaranteeing formally verified exploration. Second, we propose a new solution to this problem that combines ideas from RL and formal methods, and we show that our method has convergence guarantees. Third, we present promising experimental results for our method in the continuous control setting.

2 Preliminaries

Safe Exploration. We formulate our problem in terms of a Markov Decision Process (MDP) that has the standard probabilistic dynamics, as well as a worst-case dynamics that is used for verification. Formally, such an MDP is a structure $M = (\mathcal{S}, \mathcal{A}, P, c, \gamma, p_0, P^{\#}, \mathcal{S}_U)$. Here, $\mathcal{S}$ is a set of environment states; $\mathcal{A}$ is a set of agent actions; $P(s' \mid s, a)$ is a probabilistic transition function; $c : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a state-action cost function; $0 < \gamma < 1$ is a discount factor; $p_0(s)$ is an initial state distribution with support $\mathcal{S}_0$; $P^{\#} : \mathcal{S} \times \mathcal{A} \to 2^{\mathcal{S}}$ is a deterministic function that defines *worst-case bounds* on the environment behavior; and $\mathcal{S}_U$ is a designated set of *unsafe states* that the learner must always avoid. Because our focus is on continuous domains, we assume that $\mathcal{S}$ and $\mathcal{A}$ are real vector spaces. The function $P^{\#}$ is assumed to be available in closed form to the learner; because it captures worst-case dynamics, we require that $\operatorname{supp}(P(s' \mid s, a)) \subseteq P^{\#}(s, a)$ for all s, a. In general, the method for obtaining $P^{\#}$ will depend on the problem under consideration. In Section 4 we explain how we generate these worst case bounds for our experimental environments.

A *policy* for M is a (stochastic) map $\pi : \mathcal{S} \to \mathcal{A}$ that determines which action the agent should take in a given state. Each policy π induces a probability distribution on the cost c_i at each time step i. The aggregate cost of a policy π is $J(\pi) = \mathrm{E}[\sum_i \gamma^i c_i]$, where c_i is the cost at the i-th time step.

For a set $S \subseteq \mathcal{S}$, we define the set of states $\operatorname{reach}_i(\pi, S)$ that are reachable from S in i steps under worst-case dynamics:

$$\operatorname{reach}_1(\pi, S) = \bigcup\nolimits_{s \in S, a \in \operatorname{supp}(\pi(\cdot \mid s))} P^{\#}(s, a) \qquad \operatorname{reach}_{i+1}(\pi, S) = \operatorname{reach}_1(\pi, \operatorname{reach}_i(\pi, S)).$$

The policy π is *safe* if $(\bigcup_i \operatorname{reach}_i(\pi, \mathcal{S}_0)) \cap \mathcal{S}_U = \varnothing$. If π is safe, we write $Safe(\pi)$.

---

[1] REVEL stands for **Re**inforcement learning with **ve**rified exp**l**oration. The current implementation is available at `https://github.com/gavlegoat/safe-learning`.

Algorithm 1 Reinforcement Learning with Formally Verified Exploration (REVEL)

1: **Input:** Symbolic Policy Class $\mathcal{G}$ & Neural Policy Class $\mathcal{F}$.
2: **Input:** Initial $g_0 \in \mathcal{G}$, with the guarantee $\phi_0 \vdash Safe(g_0)$ for some ϕ_0
3: Define neurosymbolic policy class $\mathcal{H} = \{h(s) \equiv \textbf{if } P^{\#}(s, f(s)) \subseteq \phi \textbf{ then } f(s) \textbf{ else } g(s)\}$
4: **for** $t = 1, \ldots, T$ **do**
5: $\quad h_t \leftarrow \text{LIFT}_{\mathcal{H}}(g_t, \phi_t)$ *//lifting the new symbolic policy and proof into the blended space*
6: $\quad h_t \leftarrow \text{UPDATE}_{\mathcal{F}}(h_t, \eta)$ *//policy gradient in neural policy space with learning rate η*
7: $\quad (g_{t+1}, \phi_{t+1}) \leftarrow \text{PROJECT}_{\Pi}(h_t)$ *//synthesis of safe symbolic policy and corresponding invariant*
8: **end for**
9: **Return:** Policy h_T

We define a *learning process* as a sequence of policies $\mathcal{L} = \pi_0, \pi_1, \ldots, \pi_m$. We assume that the initial policy π_0 in this sequence is worst-case safe. Our algorithmic objective is to discover a learning process $\mathcal{L}$ such that the final policy π_m is safe and optimal, and every intermediate policy is safe:

$$\pi_m = \underset{\pi \text{ s.t. } Safe(\pi)}{\arg\min} \; J(\pi) \tag{1}$$

$$\forall 0 \le i \le m : Safe(\pi_i). \tag{2}$$

Formal Verification. Our learning algorithm relies on an oracle for formal verification of policies. Given a policy π, such a verifier tries to construct an inductive proof of the property $Safe(\pi)$. Such a proof takes the form of an *inductive invariant*, defined as a set of states ϕ such that: (i) ϕ includes the initial states, i.e., $\mathcal{S}_0 \subseteq \phi$; (ii) ϕ is closed under the worst-case transition relation, i.e., $\text{reach}_1(\phi) \subseteq \phi$; and (iii) ϕ does not overlap with the unsafe states, i.e., $\phi \cap \mathcal{S}_U = \emptyset$. Intuitively, states s in ϕ are such that even under worst-case dynamics, MDP trajectories from s can never encounter an unsafe state. We use the notation $\phi \vdash \pi$ to indicate that policy π can be proven safe using inductive invariant ϕ.

Inductive invariants can be constructed in many ways. Our implementation uses *abstract interpretation* [9], which maintains some *abstract* state that approximates the *concrete* states which the system can reach. For example, the abstract state might be a hyperinterval in the state space of the program that defines independent bounds on each state variable. Critically, this abstract state is an *overapproximation*, meaning that while the abstract state may include states which are not actually reachable, it will always include *at least* every reachable state. By starting with an abstraction of the initial states and using abstract interpretation to propagate this abstract state through the environment transitions and the policy, we can obtain an abstract state which includes all of the reachable states of the system (that is, we compute approximations of $\text{reach}_i(\mathcal{S}_0)$ for increasing i). Then if this abstract state does not include any unsafe states, we can be sure that none of the unsafe states are reachable by any concrete trajectory of the system either.

3 Learning Algorithm

Our learning method is a functional mirror descent in policy space, based on approximate gradients, similar to PROPEL [29]. The algorithm relies on two policy classes $\mathcal{G}$ and $\mathcal{H}$, with $\mathcal{G} \subseteq \mathcal{H}$.

The class $\mathcal{G}$ comprises the policies that we use as shields. These policies are safe and can be efficiently certified as such. Because automatic verification works better on functions that belong to certain restricted classes and are represented in compact, symbolic forms, we assume some syntactic restrictions on our shields. The specific restrictions depend on the power of the verification oracle; we describe the choices made in our implementation in Section 3.1.

The larger class $\mathcal{H}$ consists of neurosymbolic policies. Let $\mathcal{F}$ be a predefined class of neural policies. We assume that each shield in $\mathcal{G}$ can also be expressed as a policy in $\mathcal{F}$, i.e., $\mathcal{G} \subseteq \mathcal{F}$. Policies $h \in \mathcal{H}$ are of the form:

$$h(s) = \textbf{if } (P^{\#}(s, f(s)) \subseteq \phi) \textbf{ then } f(s) \textbf{ else } g(s)$$

where $g \in \mathcal{G}$, $f \in \mathcal{F}$, and ϕ is an inductive invariant that establishes $Safe(g)$. We commonly denote a policy h as above by the notation (g, ϕ, f).

The "true" branch in the definition of h represents the normal mode of the policy. The condition $P^{\#}(s, f(s)) \subseteq \phi$ is the safety monitor. If this condition holds, then the action $f(s)$ is safe, as it can

---

Algorithm 2 Implementation of $\text{PROJECT}_{\mathcal{G}}$

1: **Input:** A neurosymbolic policy $h = (g, \phi, f)$ where $g = [(g_1, \chi_1), \ldots, (g_n, \chi_n)]$
2: $g^* \leftarrow g$
3: **for** $t = 1, \ldots, T$ **do**
4:     $\psi \leftarrow \text{CUTTINGPLANE}(\chi_i)$ for heuristically selected i
5:     $g_i^1 \leftarrow \text{IMITATESAFELY}(f, g_i, \chi_i \wedge \psi); \qquad g_i^2 \leftarrow \text{IMITATESAFELY}(f, g_i, \chi_i \wedge \neg\psi)$
6:     $g' \leftarrow \text{SPLIT}(g, i, (g_i^1, \chi_i \wedge \psi), (g_i^2, \chi_i \wedge \neg\psi))$
7:     **if** $D(g', h) < D(g^*, h)$ **then**
8:         $g^* \leftarrow g'$
9:     **end if**
10: **end for**
11: $\phi^* \leftarrow \text{SAFESPACE}(g^*)$
12: **return** (g^*, ϕ^*)

---

only lead to states in ϕ (which does not overlap with the unsafe states). If the condition does not hold, then f can violate safety, and the shield g is executed in its place. In either case, h is safe. As for updates to h, we do not assume that the policy gradient $\nabla_{\mathcal{H}} J(h)$ in the space $\mathcal{H}$ exists, and approximate it by the gradient $\nabla_{\mathcal{F}} J(h)$ in the space $\mathcal{F}$ of neural policies.

We sketch our learning procedure in Algorithm 1. The algorithm starts with a (manually constructed) shield $g_0 \in \mathcal{G}$ and a corresponding invariant ϕ_0, then iteratively performs the following steps.

LIFT$_{\mathcal{H}}$. This step takes as input a shield $g \in \mathcal{G}$ and its accompanying invariant ϕ, and constructs the policy $(g, \phi, g) \in \mathcal{H}$. Note that the neural component of this policy is just the input shield g (in a neural representation). In practice, to construct this component, we can train a randomly initialized neural network to imitate g, using an algorithm such as DAGGER [26]. Because the safety of any policy (g, ϕ, f') only depends on g and ϕ, this step is safe.

UPDATE$_{\mathcal{F}}$. This procedure performs a series of gradient updates to a neurosymbolic policy $h = (g, \phi, f)$. As mentioned earlier, this step uses the approximate policy gradient $\nabla_{\mathcal{F}} J(h)$. This means that after an update, the new policy is $(g, \phi, f - \eta\nabla_{\mathcal{F}} J(h))$, for a suitable learning rate η. As the update does not change g and ϕ, the new policy is provably safe. Also, we show later that, under certain assumptions, the regret introduced by our approximation of the gradient is bounded.

PROJECT$_{\mathcal{G}}$. This procedure implements the projection operation of mirror descent. Given a neurosymbolic policy $h = (g, \phi, f)$, the procedure computes a policy $g' \in \mathcal{G}$ that satisfies $g' = \arg\min_{g'' \in \mathcal{G}} D(g'', (g, \phi, f))$ for some Bregman divergence D. Along with g', we compute an invariant ϕ' such that $\phi' \vdash Safe(g')$.

The computation of g' can be naturally cast as an imitation learning task with respect to the demonstration oracle (g, ϕ, f). Prior work [29, 30] has given heuristic solutions to this problem for the case when g'' obeys certain syntactic constraints. In our setting, we have an additional semantic requirement: g'' must be provably safe. How to solve this problem depends on the precise definition of the class of shields $\mathcal{G}$. Section 3.1 sketches the approach to this problem used in our implementation.

3.1 Instantiation with Piecewise Linear Shields

Any attempt to implement REVEL must start by choosing a class $\mathcal{G}$ of shields. Policies in $\mathcal{G}$ should be sufficiently expressive to allow for good learning performance but also facilitate verification. In our implementation, we choose $\mathcal{G}$ to comprise *deterministic, piecewise linear policies* of the form:

$$g(s) = \begin{cases} g_1(s) & \text{if } \chi_1(s) \\ g_2(s) & \text{if } \chi_2(s) \wedge \neg\chi_1(s) \\ \ldots \\ g_n(s) & \text{if } \chi_n(s) \wedge (\bigwedge_{1 \leq i < n} \neg\chi_i(s)), \end{cases}$$

where $\chi_1, \ldots, \chi_n$ are linear predicates that *partition* the state space, and each g_i is a linear function. We represent $g(s)$ as a list of pairs (g_i, χ_i). We refer to the subpart of the state space defined by $\chi_i \wedge \bigwedge_{j=1}^{i-1} \neg\chi_j$ as the *region* for linear policy g_i and denote this region by $\text{Region}(g_i)$.

Now we sketch our implementation of Algorithm 1. Since the LIFT$_\mathcal{H}$ and UPDATE$_\mathcal{F}$ procedures are agnostic to the choice of $\mathcal{G}$, we focus on PROJECT$_\mathcal{G}$, which seeks to find a shield g at minimum imitation distance $D(g, h)$ from a given $h \in \mathcal{H}$.

Our implementation of this operation is the iterative procedure in Algorithm 2. Here, we start with an input policy $h = (g, \phi, f)$. In each iteration, we identify a component g_i with region χ_i, then perform the following steps: (i) Sample a *cutting plane* that creates a more fine-grained partitioning of the safe region, by splitting the region χ_i into two new regions χ_i^1 and χ_i^2. (ii) For each new region χ_i^j, use a subroutine IMITATESAFELY to construct a safe linear policy g_i^j (and a corresponding invariant) that minimizes $D(g_i^j, h)$ within the region χ_i^j. (iii) Replace (g_i, χ_i) by the two new components, leading to the creation of a new, refined shield g'. The procedure ends by returning the most optimal shield g' (and an invariant obtained by combining the invariants of the g_i^j-s) constructed through this process.

Now we sketch IMITATESAFELY, which constructs safe and imitation-loss-minimizing linear policies. By collecting state-action pairs using DAGGER [26], we reduce the procedure's objective to a series of constrained supervised learning problems. Each of these problems is solved using a projected gradient descent (PGD) that alternates between gradient updates to a linear policy and projections into the set of safe linear policies. Critically, the constraint imposed on each of these optimization problems is constructed such that (i) the resulting policy is provably safe and (ii) the projection for the PGD algorithm is easy to compute. In our implementation these constraints take the form of a hyperinterval in the parameter space of the linear policies. We can then use abstract interpretation [9], a common framework for program verification, to prove that every controller within a particular hyperinterval behaves safely. For more details on IMITATESAFELY, see the supplementary material.

3.2 Theoretical Analysis

The REVEL approach introduces two new sources of error over standard mirror descent. First, we approximate the gradient $\nabla_\mathcal{H}$ by $\nabla_\mathcal{F}$, which introduces bias. Second, our projection step may be inexact. Prior work [29] has studied methods for implementing the projection step with bounded error. Here, we bound the bias in the gradient approximation under some simplifying assumptions, and use this result to prove a regret bound on the final shield that our method converges on. We define a safety indicator Z which is zero whenever the shield is invoked and one otherwise. We assume:

1. $\mathcal{H}$ is a vector space equipped with an inner product $\langle \cdot, \cdot \rangle$ and induced norm $\|h\| = \sqrt{\langle h, h \rangle}$,
2. J is convex in $\mathcal{H}$, and ∇J is L_J-Lipschitz continuous on $\mathcal{H}$,
3. $\mathcal{H}$ is bounded (i.e., $\sup\{\|h - h'\| \mid h, h' \in \mathcal{H}\} < \infty$),
4. $\mathbb{E}[1 - Z] \leq \zeta$, i.e., the probability that the shield is invoked is bounded above by ζ,
5. the bias introduced in the sampling process is bounded by β, i.e., $\|\mathbb{E}[\widehat{\nabla}_\mathcal{F} \mid h] - \nabla_\mathcal{F} J(h)\| \leq \beta$, where $\widehat{\nabla}_\mathcal{F}$ is the estimated gradient
6. for $s \in \mathcal{S}$, $a \in \mathcal{A}$, and policy $h \in \mathcal{H}$, if $h(a \mid s) > 0$ then $h(a \mid s) > \delta$ for some fixed $\delta > 0$.

Intuitively, this last assumption amounts to cutting of the tails of the distribution so that no action can be arbitrarily unlikely. Now, let the variance of the gradient estimates be bounded by σ^2, and assume the projection error $\|g_t - g_t^*\| \leq \epsilon$ where g_t^* is the exact projection of a neurosymbolic policy onto $\mathcal{G}$ and g_t is the computed projection. Let R be an α-strongly convex and L_R-strongly smooth regularizer. Then the bias of our gradient estimate is bounded by Lemma 1 and the expected regret bound is given by Theorem 1.

Lemma 1. *Let γ be the diameter of $\mathcal{H}$, i.e., $\gamma = \sup\{\|h - h'\| \mid h, h' \in \mathcal{H}\}$. Then the bias incurred by approximating $\nabla_\mathcal{H} J(h)$ with $\nabla_\mathcal{F} J(h)$ and sampling is bounded by*

$$\left\| \mathbb{E}\left[\widehat{\nabla}_t \mid h \right] - \nabla_\mathcal{H} J(h) \right\| = O(\beta + L_j \zeta).$$

Theorem 1. *Let $g_1, \ldots, g_T$ be a sequence of shields in $\mathcal{G}$ returned by REVEL and let g^* be the optimal programmatic policy. Choosing a learning rate $\eta = \sqrt{\frac{1}{\sigma^2}\left(\frac{1}{T} + \epsilon\right)}$ we have the expected regret over T iterations:*

$$\mathbb{E}\left[\frac{1}{T}\sum_{i=1}^{T} J(g_i) \right] - J(g^*) = O\left(\sigma\sqrt{\frac{1}{T} + \epsilon} + \beta + L_J\zeta \right)$$

This theorem matches the expectation that when a blended policy $h = (g, \phi, f)$ is allowed to take more actions without the shield intervening (i.e., ζ decreases), the regret bound is decreased. Intuitively, this is because when we use the shield, the action we take does not depend on the neural network f, so the learner does not learn anything useful. However if h is using f to choose actions, then we have unbiased gradient information as in standard RL.

4   Experiments

Now we present our empirical evaluation of REVEL. We investigate two questions: (1) How much safer are REVEL policies compared to state-of-the-art RL techniques that lack worst-case safety guarantees? What is the performance penalty for this increased safety? (2) Does REVEL offer significant performance gains over prior verified exploration approaches based on static shields[11, 3]?

To answer these questions, we compared REVEL against three baselines: (1) Deep Deterministic Policy Gradients (DDPG) [22]; (2) Constrained policy optimization (CPO) [1]; and (3) a variant of REVEL that never updates the user-provided shield. Of these, CPO is designed for safe exploration and takes into account a safety cost function. For DDPG, we engineered a reward function that has a penalty for safety violations. Details of hyperparameters that we used appear in the Appendix.

Our experiments used 10 benchmarks that include classic control problems, robotics applications, and benchmarks from prior work [11]. For each of these environments, we hand-constructed a worst-case, piecewise linear model of the dynamics. These models are based on the physics of the environment and use non-determinism to approximate nonlinear functions. For example, some of our benchmarks include trigonometric functions which cannot be represented linearly. In these cases, we define piecewise linear upper and lower bounds to the trigonometric functions. These linear approximations are necessary to make verification feasible. Each benchmark also includes a bounded-time safety property which should hold at all times during training.

Performance. First, we compare the policies learned using REVEL against policies learned using the baselines in terms of their cost (lower is better). Figures 1 and 2 show the cost over time of the policies during training. The results suggest that:

- The performance of REVEL is competitive with (or even better than) DDPG for 7 out of the 10 benchmarks. REVEL achieves significantly better reward than DDPG in the "car-racing" benchmark, and reward is only slightly worse for 2 benchmarks.
- REVEL has better performance than CPO on 4 out of the 10 benchmarks and only performs slightly worse on 2. Furthermore, the cost incurred by CPO is significantly worse on 2 benchmarks (noisy-road and car-racing).
- REVEL outperforms the static shielding approach on 4 out of 10 benchmarks. Furthermore, the difference is very substantial on two of these benchmarks (noisy-road and mountain-car).

REVEL does induce substantial overhead in terms of computational cost. The cost of the network updates and shield updates for each benchmark are shown in Table 1 along with the percentage of the total time spent in shield synthesis. The "acc" and "pendulum" benchmarks stand out as having very fast shield updates. For these two benchmarks the safety properties are relatively simple, so the verification engine is able to come up with safe shields more quickly. Otherwise, REVEL spends the majority of its time (87% on average) on shield synthesis.

Safety. To validate whether the safety guarantee provided by REVEL is useful, we consider how DDPG and CPO behave during training. Specifically, Table 2 shows the average number of safety violations per run for DDPG and CPO. As we can see from this table, DDPG and CPO both exhibit safety violations in 8 out of the 10 benchmarks. In Figure 3, we show how the number of violations varies throughout the training process for a few of the benchmarks. The remaining plots are left to the supplementary material.

Qualitative Assessment. To gain intuition about the difference between policies that REVEL and the baselines compute, we consider trajectories from the trained policies for two of our benchmarks that are easy to visualize. Figure 4 shows the trajectories taken by each of the policies for the obstacle2 benchmark. In this environment, the policy starts in the lower left corner, and the goal is to move to the green circle in the upper right. However, the red box in the middle is unsafe. As we can see from Figure 4, all of the policies have learned to go around the unsafe region in the center. However

(a) mountain-car

(b) road

(c) road-2d

(d) noisy-road

(e) noisy-road-2d

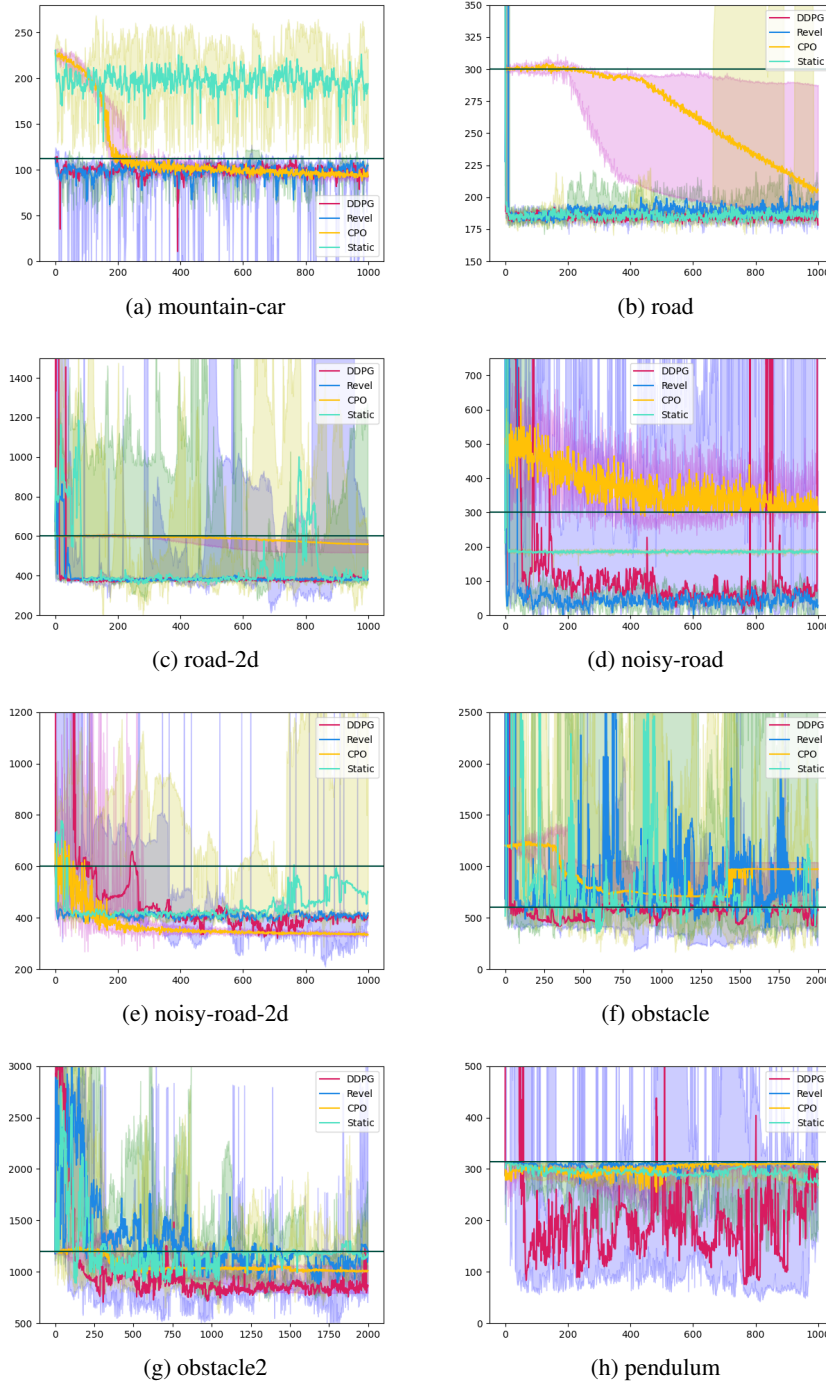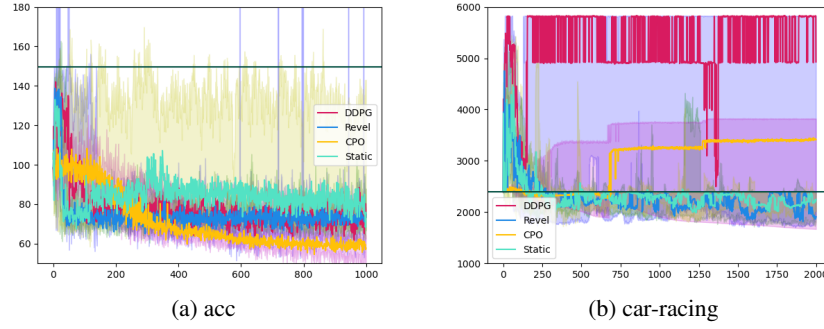(f) obstacle

(g) obstacle2

(h) pendulum

Figure 1: Training performance comparison on our benchmarks. The y-axis represents the Cost $J(\pi)$ and the x-axis gives the number of training episodes.

DDPG has not reinforced this behavior enough and still enters the unsafe region at the corner. By contrast, the statically shielded policy manages to avoid the region, but there is a very clear bend in its trajectory where the shield has to step in. Revel avoids the unsafe region while maintaining a smooth trajectory throughout. In this case, CPO also learns to avoid the unsafe region and go to the goal. (Because the environment is symmetrical, there is no significance to the CPO curve going up first and then right.)

(a) acc

(b) car-racing

Figure 2: Training performance comparison (continued). The y-axis represents the Cost $J(\pi)$ and the x-axis gives the number of training episodes.

Table 1: Training time in seconds for network and shield updates.

Benchmark	Network update (s)	Shield update (s)	Shield percentage
mountain-car	1900	5315	73.7%
road	954	9401	90.8%
road-2d	1015	19492	95.1%
noisy-road	962	12793	93.0%
noisy-road-2d	935	25514	96.5%
obstacle	4332	27818	86.5%
obstacle2	4365	21661	83.2%
pendulum	1292	113	8.0%
acc	1097	56	4.9%
car-racing	4361	15892	78.5%

Table 2: Safety violations.

Benchmark	DDPG	CPO
mountain-car	0	3.6
road	0	0
road-2d	113.4	70.8
noisy-road	1130.4	8526.4
noisy-road-2d	107.4	0
obstacle	12.4	1.0
obstacle2	96	118.6
pendulum	92.4	9906
acc	4	673
car-racing	4956.2	22.4

Figure 5 shows trajectories for "acc", which models an adaptive cruise control system where the goal is to follow a lead car as closely as possible without crashing into it. The lead car can apply an acceleration to itself at any time. The x-axis shows the distance to the lead car while the y-axis shows the relative velocities of the two cars. Here, all three trajectories start by accelerating to close the gap to the lead car before slowing down again. The statically shielded (and most conservative) policy is the first to slow down. The DDPG and CPO policies fail to slow down soon enough or quickly enough and crash into the lead car (the red region on the right side of the figure). In contrast, the REVEL policy can more quickly close the gap to the lead car and slow down later while still avoiding a crash.
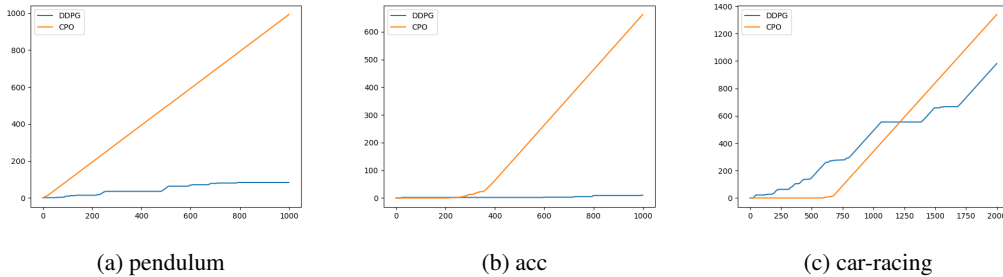

(a) pendulum

(b) acc

(c) car-racing

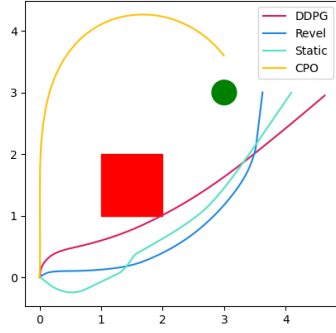Figure 3: Cumulative safety violations during training.
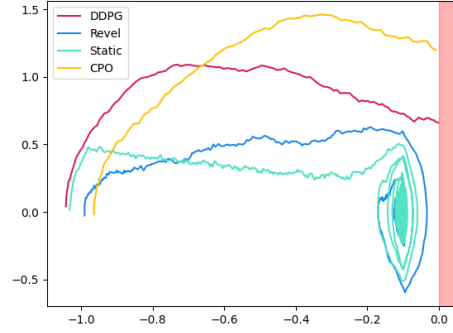
8

Figure 4: Trajectories for obstacle2.

Figure 5: Trajectories for acc.

5   Related Work

There is a growing literature on safety in RL [14]. Approaches here can be classified on basis of whether safety is guaranteed during learning or deployment. REVEL, and, for example, CPO [1], were designed to enforce safety during training. Another way to categorize approaches is by whether their guarantees are probabilistic (or in expectation) or worst-case. Most approaches [23, 7, 1, 7] are in the former category; however, REVEL and prior approaches based on verified monitors [3, 11, 12] are in the latter camp. Now we discuss in more detail three especially related threads of work.

Safety via Shielding. These approaches rely on a definition of error states or fatal transitions to guarantee safety and have been used extensively in both RL and control theory [2, 3, 8, 11, 12, 16, 24, 32]. Our approach follows this general framework, but crucially introduces a mechanism to improve the shielded-policy during training. This is achieved by projecting the neural policy onto the shielded policy space. The idea of synthesizing a shield to imitate a neural policy has been explored in recent work [5, 32]. However, these approaches only generated the shield after training, so there are no guarantees about safety during training.

Formal Verification of Neural Networks. There is a growing literature on the verification of worst-case properties of neural networks [4, 15, 18, 19, 31]. In particular, a few recent papers [17, 27] target the verification of neural policies for autonomous agents. However, performing such verification inside a learning loop is computationally infeasible – in fact, state-of-the-art techniques failed to verify a single network from our benchmarks within half an hour.

Safety via Optimization Constraints. Many recent approaches to safe RL rely on specifying safety constraints as an additional cost function in the optimization objective [1, 6, 10, 20, 21]. These approaches typically provide safety up to a certain threshold by requiring that the additional cost function is kept below a certain constant. In contrast, our approach is suitable for use cases in which safety violations are completely unacceptable and where provable guarantees are required.

6   Conclusion

We have presented REVEL, an RL framework that permits formally verified exploration while supporting continuous action spaces and contemporary learning algorithms. Our key innovation is to cast the verified RL problem as a form of mirror descent that uses a verification-friendly symbolic policy representation along with a neurosymbolic policy representation that benefits learning.

One limitation of this work is its assumption of a fixed worst-case model of the environment. Allowing this model to be updated as learning progresses [12] is a direction for future work. The development of incremental verification techniques to further allay the cost of repeated verification is another natural direction. Progress on such verification techniques can potentially allow the use of more expressive classes of shields, which, in turn, can boost the learner's overall performance.

Broader Impact

In the recent past, reinforcement learning has seen numerous advances and found applications in safety-critical settings. System failures in this setting can result in significant loss of property or even loss of life. This work takes a step towards solving this problem by guaranteeing that RL agents do not violate safety properties.

As with any safety-related work, the consequences of failure or misuse of this technique can be severe. Specifically, there is a risk that a user might assume that their system is guaranteed safe when this is not the case (for example, if the user fails to adequately specify the environment or safety property). Writing correct safety specifications is known to be hard, so inexpert users may feel an unwarranted sense of security. While misuse of the tool carries great risk, proper use can confer substantial advantages. In particular, it may allow the benefits of RL to be brought to domains, such as robotics and autonomous vehicles, where failure has a very high cost.

Funding Acknowledgment

References

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 22–31. PMLR, 2017.

[2] Anayo K. Akametalu, Shahab Kaynama, Jaime F. Fisac, Melanie Nicole Zeilinger, Jeremy H. Gillula, and Claire J. Tomlin. Reachability-based safe learning with gaussian processes. In *53rd IEEE Conference on Decision and Control, CDC 2014, Los Angeles, CA, USA, December 15-17, 2014*, pages 1424–1431, 2014.

[3] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 2669–2678. AAAI Press, 2018.

[4] Greg Anderson, Shankara Pailoor, Isil Dillig, and Swarat Chaudhuri. Optimization and abstraction: a synergistic approach for analyzing neural network robustness. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 731–744, 2019.

[5] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Advances in Neural Information Processing Systems*, pages 2494–2504, 2018.

[6] Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in neural information processing systems*, pages 908–918, 2017.

[7] Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. A Lyapunov-based approach to safe reinforcement learning. In *Advances in neural information processing systems*, pages 8092–8101, 2018.

[8] Yinlam Chow, Ofir Nachum, Edgar A. Duéñez-Guzmán, and Mohammad Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 8103–8112.

[9] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM, 1977.

[10] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe Exploration in Continuous Action Spaces. January 2018.

[11] Nathan Fulton and André Platzer. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[12] Nathan Fulton and André Platzer. Verifiably safe off-model reinforcement learning. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 413–430. Springer, 2019.

[13] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[14] Javier Garcıa and Fernando Fernandez. A Comprehensive Survey on Safe Reinforcement Learning. page 44.

[15] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2018.

[16] Jeremy H. Gillula and Claire J. Tomlin. Guaranteed safe online learning via reachability: tracking a ground target using a quadrotor. In *IEEE International Conference on Robotics and Automation, ICRA 2012, 14-18 May, 2012, St. Paul, Minnesota, USA*, pages 2723–2730, 2012.

[17] Radoslav Ivanov, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 169–178, 2019.

[18] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[19] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. The marabou framework for verification and analysis of deep neural networks. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 443–452. Springer, 2019.

[20] Hoang M Le, Cameron Voloshin, and Yisong Yue. Batch policy learning under constraints. In *International Conference on Machine Learning (ICML)*, 2019.

[21] Xiao Li and Calin Belta. Temporal Logic Guided Safe Reinforcement Learning Using Control Barrier Functions. *arXiv:1903.09885 [cs, stat]*, March 2019. arXiv: 1903.09885.

[22] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

[23] Teodor Mihai Moldovan and Pieter Abbeel. Safe exploration in markov decision processes. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress, 2012.

[24] Theodore J. Perkins and Andrew G. Barto. Lyapunov design for safe reinforcement learning. *J. Mach. Learn. Res.*, 3:803–832, 2002.

[25] Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking Safe Exploration in Deep Reinforcement Learning. 2019.

[26] Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, pages 627–635, 2011.

[27] Xiaowu Sun, Haitham Khedr, and Yasser Shoukry. Formal verification of neural network controlled autonomous systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 147–156, 2019.

[28] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[29] Abhinav Verma, Hoang Le, Yisong Yue, and Swarat Chaudhuri. Imitation-projected programmatic reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 15726–15737, 2019.

[30] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pages 5052–5061, 2018.

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

[32] He Zhu, Zikang Xiong, Stephen Magill, and Suresh Jagannathan. An inductive synthesis framework for verifiable reinforcement learning. *CoRR*, abs/1907.07273, 2019.

A   Safely Imitating a Neural Policy

In this section, we describe our projection algorithm for piecewise linear policies in more detail. Algorithm 2 defines this operation at a high-level, but leaves out some of the details of the IMITATE-SAFELY subroutine. The role of IMITATESAFELY is to learn a linear policy which is provably safe on some region and behaves as similarly to the neural controller as possible on that region. Since linear policies are differentiable, we adopt a projected gradient descent approach. To formalize this, we note that a linear policy g is just a matrix in $\mathbb{R}^{n \times m}$ where n is the dimension of the action space and m is the dimension of the state space. We will use θ_g to refer to a vector in $\mathbb{R}^{nm}$ parameterizing g.

Algorithm 3 Safely imitating a network using a given starting point and partition.

1: **Input:** A neural policy f, a region χ, and a linear policy g.
2: $g^* \leftarrow g$
3: **while** * **do**
4:     $S \leftarrow \text{COMPUTESAFEREGION}(g^*, \chi)$
5:     $g^* \leftarrow g^* - \alpha \nabla D(g^*, N)$
6:     $g^* \leftarrow proj_S(g^*)$
7: **end while**
8: **return** g^*

Now our safe imitation algorithm is described in Algorithm 3. In each iteration, we first compute a safe region in the parameter space of g^* over the region χ. This is done by starting with a region bigger than the gradient step size and then iteratively searching for unsafe controllers and trimming the region to remove them. The returned region $S \subseteq \mathbb{R}^{n \times m}$ contains g^* and only contains safe policies over the region χ. This trimming process continues until S can be verified using abstract interpretation [9]. In our implementation S represents an interval in $\mathbb{R}^{nm}$ constraining θ_g. Next, we take a gradient step according to the imitation loss D. For example D may be computed using a DAgger-like algorithm to gather a dataset for supervised learning. Finally, we project g^* into the safe region S computed earlier. Specifically, this means projecting θ_{g^*} into the region of $\mathbb{R}^{nm}$ represented by S. Notice that since we project back into S after each iteration, the policy returned by IMITATESAFELY is known to be safe on χ.

Intuitively, recomputing S at each iteration allows the controller to learn behavior which is more different from the starting point g than would otherwise be possible. This is because computing a safe region involves abstracting the behavior of the system and in general it is intractable to compute the entire safe of safe policies in advance. Recomputing the safe space using the current controller at each step means we only need to prove the safety of a relatively small piece of the policy space local to the current controller. Specifically, if we can verify a region at least as large as one gradient step then the gradient descent procedure is unconstrained for that step. By repeating this process at each step, we only end up needing to verify a thin strip of policies surrounding the trajectory the gradient descent algorithm takes through the policy space.

B   Theoretical Analysis

Here we provide proofs of the theoretical results from Section 3.2 and extend the discussion of a few theoretical issues.

Recall from Section 3.2 that we require the policy space $\mathcal{H}$ to be a vector space equipped with an inner product $\langle \cdot, \cdot \rangle$ inducing a norm $\|h\| = \sqrt{\langle h, h \rangle}$. Addition and scalar multiplication are defined in the standard way, i.e., $(\lambda u + \kappa v)(s) = \lambda u(s) + \kappa v(s)$. The cost functional of a policy u is defined as $J(u) = \int_{\mathcal{S}} c(s, u(s)) \mathrm{d}\mu^u(s)$ where μ^u is the state distribution induced by u. We assume that $\mathcal{G}$ and $\mathcal{F}$ are subspaces of $\mathcal{H}$ so that there is a well-defined notion of distance between policies in these classes. Additionally, notice that while a policies in $\mathcal{G}$ may not be differentiable in terms of their programmatic representation, they may still be differentiable when viewed as points in the ambient space $\mathcal{H}$. We will assume $\mathcal{H}$ is parameterizable by a vector in $\mathbb{R}^N$ for some N.

We will make use of a few standard notions from functional analysis, restated here for convenience:

Definition 1. *(Strong convexity) A differentiable function R is α-strongly convex w.r.t. a norm $\| \cdot \|$ if $R(y) \geq R(x) + \langle \nabla R(x), y - x \rangle + \frac{\alpha}{2} \|y - x\|^2$.*

Definition 2. *(Lipschitz continuous gradient smoothness) A differentiable function R is L_R-strongly smooth w.r.t. a norm $\|\cdot\|$ if $\|\nabla R(x) - \nabla R(y)\|_* \leq L_R\|x - y\|$.*

Definition 3. *(Bregman divergence) For a strongly convex regularizer R, $D_R(x, y) = R(x) - R(y) - \langle \nabla R(y), x - y \rangle$ is the Bregman divergence between x and y. Note that D_R is not necessarily symmetric.*

With these preliminaries, we can now prove Theorem 1 from Section 3.2. The high-level strategy for this proof will be to prove Lemma 1, and then combine this result with a more general regret bound from [29]. First we restate the general theorem below. Let R be an α-strongly convex and L_R-smooth functional w.r.t. the norm $\|\cdot\|$ on $\mathcal{H}$. Additionally let $\nabla_{\mathcal{H}}$ be a Fréchet gradient on $\mathcal{H}$. Then our algorithm can be described as follows: start with $g_0 \in \mathcal{G}$ (provided by the user) then for each iteration t:

1. Compute a noisy estimate of the gradient $\widehat{\nabla}J(g_{t-1}) \approx \nabla J(g_{t-1})$.

2. Update in $\mathcal{H}$: $\nabla R(h_t) = \nabla R(h_{t-1}) - \eta\widehat{\nabla}J(g_{t-1})$.

3. Perform an approximate projection $g_t = \text{proj}_{\mathcal{G}}^R(h_t) \approx \arg\min_{g \in \mathcal{G}} D_r(g, h_t)$.

This procedure is approximate functional mirror descent under bandit feedback. We let D be the diameter of $\mathcal{G}$, i.e., $D = \sup\{\|g - g'\| \mid g, g' \in \mathcal{G}\}$. L_J is the Lipschitz constant of the functional J on $\mathcal{H}$. β and σ^2 are bounds on the bias and variance, respectively, of the gradient estimate in each iteration. α and L_R are the strongly convex and smooth coefficients of the functional regularizer R. Finally ϵ is the bound on the projection error with respect to the same norm $\|\cdot\|$. We will make use of the following general result:

Theorem 2. *[29] Let $g_1, \ldots, g_T$ be a sequence of programmatic policies returned by REVEL and g^* be the optimal programmatic policy. We have the expected regret bound*

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} J(g_t)\right] - J(g^*) \leq \frac{L_R D^2}{\eta T} + \frac{\epsilon L_R D}{\eta} + \frac{\eta(\sigma^2 + L_J^2)}{\alpha} + \beta D.$$

In particular, choosing $\eta = \sqrt{(1/T + \epsilon)/\sigma^2}$, this simplifies to

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} J(g_t)\right] - J(g^*) = O\left(\sigma\sqrt{\frac{1}{T} + \epsilon} + \beta\right).$$

Now we restate and prove Lemma 1 from the main paper to provide a bound on the bias of our gradient estimate. Recall our definition of the immediate safety indicator Z as zero if the shield is invoked and one otherwise. Recall the assumptions from Section 3.2:

1. J is convex in $\mathcal{H}$ and ∇J is L_J-Lipschitz continuous on $\mathcal{H}$,

2. $\mathcal{H}$ is bounded (i.e., $\sup\{\|h - h'\| \mid h, h' \in \mathcal{H}\} < \infty$),

3. $\mathbb{E}[1 - Z] \leq \zeta$, i.e., the probability that the shield is invoked is bounded above by ζ,

4. the bias introduced in the sampling process is bounded by β, i.e., $\|\mathbb{E}[\widehat{\nabla}_{\mathcal{F}} \mid h] - \nabla_{\mathcal{F}}J(h)\| \leq \beta$, where $\widehat{\nabla}_{\mathcal{F}}$ is the estimated gradient

5. for $s \in \mathcal{S}$, $a \in \mathcal{A}$, and policy $h \in \mathcal{H}$, if $h(a \mid s) > 0$ then $h(a \mid s) > \delta$ for some fixed $\delta > 0$.

Under these assumptions:

Lemma. *Let D be the diameter of $\mathcal{H}$, i.e., $D = \sup\{\|h - h'\| \mid h, h' \in \mathcal{H}\}$. Then the bias incurred by approximating $\nabla_{\mathcal{H}}J(h)$ with $\nabla_{\mathcal{F}}J(h)$ and sampling is bounded by*

$$\left\|\mathbb{E}\left[\widehat{\nabla}_{\mathcal{F}} \mid h\right] - \nabla_{\mathcal{H}}J(h)\right\| = O(\beta + L_J\zeta).$$

Proof. First, we note that $\|\mathbb{E}[\widehat{\nabla}_{\mathcal{F}} \mid h] - \nabla_{\mathcal{H}}J(h)\| \leq \|\mathbb{E}[\widehat{\nabla}_{\mathcal{F}} \mid h] - \nabla_{\mathcal{F}}J(h)\| + \|\nabla_{\mathcal{F}}J(h) - \nabla_{\mathcal{H}}J(h)\|$. We have already assumed that the first term is bounded by β, so we will proceed to bound the second term.

Let $h = (g, \phi, f)$ be a policy in $\mathcal{H}$. By the policy gradient theorem [28], we have that

$$\nabla_{\mathcal{F}} J(h) = \mathbb{E}_{s \sim \rho_h, a \sim h} \left[\nabla_{\mathcal{F}} \log h(a \mid s) Q^h(s, a) \right] \tag{3}$$

where ρ_h is the state distribution induced by h and Q^h is the long-term expected reward from a state s and action a. We will omit the distribution subscript in the remainder of the proof for convenience. Now note that if Z is one, then then $h(a \mid s) = f(a \mid s)$, so that in particular

$$\nabla_{\mathcal{F}} \log h(a \mid s) Q^h(s, a) = \nabla_{\mathcal{H}} \log h(a \mid s) Q^h(s, a).$$

On the other hand, if Z is zero, then $h(a \mid s)$ is independent of f, and so we have

$$\nabla_{\mathcal{F}} \log h(a \mid s) Q^h(s, a) = 0.$$

Thus, we can rewrite Equation 3 as

$$\begin{aligned} \nabla_{\mathcal{F}} J(h) &= \mathbb{E} \left[Z \nabla_{\mathcal{H}} \log h(a \mid s) Q^h(s, a) \right] \\ &= \mathbb{E}[Z] \mathbb{E} \left[\nabla_{\mathcal{H}} \log h(a \mid s) Q^h(s, a) \right] + \text{Cov}(S, \nabla_{\mathcal{H}} \log h(a \mid s) Q^h(s, a)) \\ &= \mathbb{E}[Z] \nabla_{\mathcal{H}} J(h) + \text{Cov}(S, \nabla_{\mathcal{H}} \log h(a \mid s) Q^h(s, a)). \end{aligned} \tag{4}$$

Note that the covariance term is a vector where the i'th component is the covariance between Z and the i'th component of the gradient $\nabla_{\mathcal{H}}^i$. Then for each i, by Cauchy-Schwarz we have

$$|\text{Cov}(Z, \nabla_{\mathcal{H}}^i \log h(a \mid s) Q^h(s, a))| \leq \sqrt{\text{Var}(Z) \text{Var}(\nabla_{\mathcal{H}}^i \log h(a \mid s) Q^h(s, a))}.$$

Since $Z \in \{0, 1\}$ we must have $0 \leq \text{Var}[Z] \leq 1$ so that

$$|\text{Cov}(Z, \nabla_{\mathcal{H}}^i \log h(a \mid s) Q^h(s, a))| \leq \sqrt{\text{Var}(\nabla_{\mathcal{H}}^i \log h(a \mid s) Q^h(s, a))}.$$

By assumption, for every state-action pair (a, s) if (a, s) is in the support of ρ_h then $h(a \mid s) > \delta$. We also have that $Q^h(s, a)$ is bounded (because J is Lipschitz on $\mathcal{H}$ and $\mathcal{H}$ is bounded). Then because the gradient of the log is bounded above by one and because $\nabla_H H$ is bounded by definition, we have $\|\nabla_{\mathcal{H}}^i \log h(a \mid s) Q^h(s, a)\|$ is bounded. Therefore by Popoviciu's inequality, $\text{Var}(\nabla_{\mathcal{H}}^i \log h(a \mid s) Q^h(s, a))$ is bounded as well. Choose $B > \text{Var}(\nabla_{\mathcal{H}}^i \log h(a \mid s) Q^h(s, a))$ for all i. Then we have $\|\text{Var}(\nabla_{\mathcal{H}} \log h(a \mid s) Q^h(s, a))\|_\infty < \sqrt{B}$, and because $\mathcal{H}$ is finite-dimensional, $\|\text{Var}(\nabla_{\mathcal{H}} \log h(a \mid s) Q^h(s, a))\| < c\sqrt{B}$ for some constant c for any norm $\| \cdot \|$.

Substituting this into Equation 4, we have

$$\|\nabla_{\mathcal{F}} J(h) - \mathbb{E}[S] \nabla_{\mathcal{H}} J(h)\| < c\sqrt{B}.$$

Then

$$\begin{aligned} \|\nabla_{\mathcal{F}} J(h) - \nabla_{\mathcal{H}} J(h)\| &\leq \|\nabla_{\mathcal{F}} J(h) - \mathbb{E}[S] \nabla_{\mathcal{H}} J(h)\| + \|\mathbb{E}[S] \nabla_{\mathcal{H}} J(h) - \nabla_{\mathcal{H}} J(h)\| \\ &< c\sqrt{B} + \|\mathbb{E}[S] \nabla_{\mathcal{H}} J(h) - \nabla_{\mathcal{H}} J(h)\|. \end{aligned}$$

By assumption, $\nabla_{\mathcal{H}} J(h)$ is Lipschitz and $\mathcal{H}$ is bounded. Let D be the diameter of $\mathcal{H}$ and recall that L_J is the Lipschitz constant of $\nabla_{\mathcal{H}} J(h)$. Choose an arbitrary $h_0 \in \mathcal{H}$ and let $J_0 = \nabla_{\mathcal{H}} J(h_0)$. Then for any policy $h \in \mathcal{H}$ we have $\|\nabla_{\mathcal{H}} J(h)\| \leq J_0 + DL_J$. Then

$$\begin{aligned} \|\mathbb{E}[Z] \nabla_{\mathcal{H}} J(h) - \nabla_{\mathcal{H}} J(h)\| &= \|(\mathbb{E}[Z] - 1) \nabla_{\mathcal{H}} J(h)\| \\ &= |\mathbb{E}[Z] - 1| \|\nabla_{\mathcal{H}} J(h)\| \\ &\leq |\mathbb{E}[Z] - 1| (J_0 + DL). \end{aligned}$$

Since Z is an indicator variable, we have $0 \leq \mathbb{E}[Z] \leq 1$ so that $|\mathbb{E}[Z] - 1| = 1 - \mathbb{E}[Z]$. Then finally we assume D is a known constant to simplify presentation, and arrive at

$$\|\nabla_{\mathcal{F}} J(h) - \nabla_{\mathcal{H}} J(h)\| < c\sqrt{B} + (1 - \mathbb{E}[Z])(J_0 + DL_J) = O(L_J \zeta)$$

and plugging this back into the original triangle inequality we have

$$\left\| \mathbb{E} \left[\widehat{\nabla}_{\mathcal{F}} \mid h \right] - \nabla_{\mathcal{H}} J(h) \right\| = O(\beta + L_J \zeta).$$

$\square$

Now Theorem 1 follows directly by plugging this bound on gradient estimate bias into Theorem 2.

C Experimental Data and Additional Results

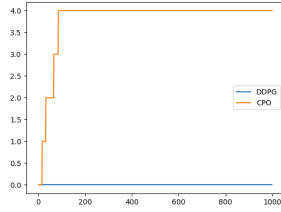In this section we provide more details about our experiments along with additional results.

First, we give a qualitative description of each benchmark:

- mountain-car is a continuous version of the classic mountain car problem. In this environment the goal is to move an underpowered vehicle up a hill by rocking back and forth in a valley to build up momentum. The safety property asserts that the car does not go over the crest of the hill on the left.

- road, road-2d, noisy-road, and noisy-road-2d are all variants of an autonomous car control problem. In each case, the car's goal is to move to a specified end position while obeying a given speed limit. The noisy variants introduce noise in the environment, while the 2d variants involve moving in two dimensions to reach the goal.

- In obstacle and obstacle2, a robot moving in 2D space must reach a goal position while avoiding an obstacle. In obstacle this obstruction is placed off to the side so it only affects the agent during exploration (but the shortest path to the goal does not intersect it). In obstacle2, the obstruction is placed between the starting position and the goal so that the policy must learn to move around it (see Figure 4).

- pendulum is a classic pendulum environment where the system must swing a pendulum up until it is vertical. The safety property in this case is a bound on the angular velocity of the pendulum.

- acc is an adaptive cruise control benchmark taken from [11] and modified to use a continuous action space. Here the goal is to follow a lead car as closely as possible without crashing into it. At each time step the lead car chooses an acceleration at random (from a truncated normal distribution) to apply to itself.

- car-racing is similar to obstacle2 except that in this case the goal is to reach a goal state on the opposite side of the obstacle and then come back. This requires the agent to complete a loop around the obstacle.
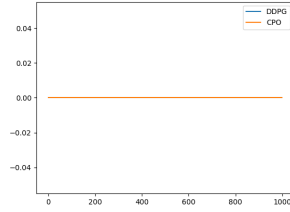
For each benchmark, we consider a bounded-time variant of the desired safety property. That is, for some fixed T we guarantee that a policy $h = (g, \phi, f)$ cannot violate the safety property within T time steps starting from any state satisfying ϕ.

For most benchmarks, we train for 100,000 environment interactions with a maximum episode length of 100. For mountain-car we use a maximum episode length of 200 and 200,000 total environment interactions. For obstacle, obstacle2, and car-racing we use an episode length of 200 with 400,000 total environment interactions. For every benchmark we synthesize five new shields at even intervals throughout training. To evaluate CPO we use the implementation provided with the Safety Gym repository [25]. To account for our safety critical benchmarks, we reduce the tolerance for safety violations in this implementation by lowering the corresponding hyper-parameter from 25 to 1. For DDPG, we use an implementation from prior work [32], which is also what we base the code for REVEL on. We ran each experiment with five independent, randomly chosen seeds. Note that the chosen number of training episodes was enough for the baselines to converge, in the sense that over the last 25 training episodes we see less than a 2% improvement in policy performance.
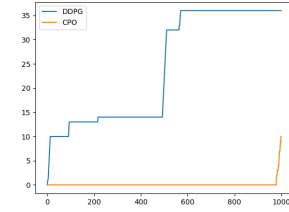
We now provide more details about the safety violations seen during training. The plots in Figure 6 show the number of safety violations over time for DDPG and CPO. This figure is the same as Figure 3 except that it shows information for every benchmark.
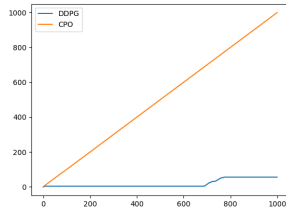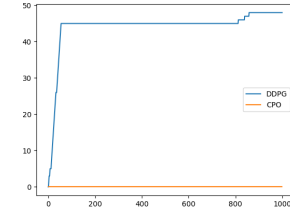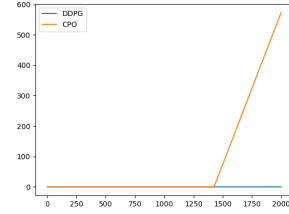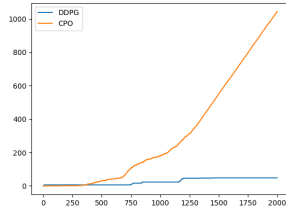
(a) mountain-car

(b) road
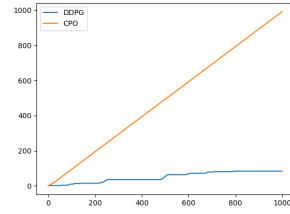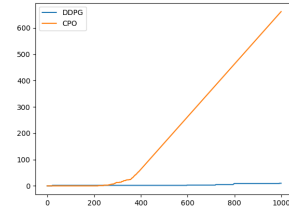
(c) road-2d

(d) noisy-road
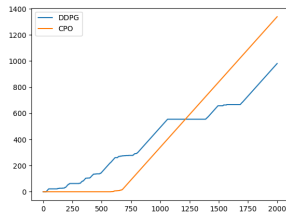
(e) noisy-road-2d

(f) obstacle

(g) obstacle2

(h) pendulum

(i) acc

(j) car-racing

Figure 6: Safety violations over time.