Verification-Guided Shielding for Deep Reinforcement Learning

Davide Corsi[1,*], **Guy Amir**[2,*], **Andoni Rodríguez**[3,4], **César Sánchez**[3], **Guy Katz**[2] **and Roy Fox**[1]

[1]University of California, Irvine, USA
[2]The Hebrew University of Jerusalem, Israel
[3]IMDEA Software Institute, Spain
[4]Universidad Politécnica de Madrid, Spain

Abstract

In recent years, Deep Reinforcement Learning (DRL) has emerged as an effective approach to solving real-world tasks. However, despite their successes, DRL-based policies suffer from poor reliability, which limits their deployment in safety-critical domains. Various methods have been put forth to address this issue by providing formal safety guarantees. Two main approaches include *shielding* and *verification*. While shielding ensures the safe behavior of the policy by employing an external online component (i.e., a "shield") that overrides potentially dangerous actions, this approach has a significant computational cost as the shield must be invoked at runtime to validate *every* decision. On the other hand, verification is an offline process that can identify policies that are unsafe, prior to their deployment, yet, without providing alternative actions when such a policy is deemed unsafe. In this work, we present *verification-guided shielding* — a novel approach that bridges the DRL reliability gap by integrating these two methods. Our approach combines both formal and probabilistic verification tools to partition the input domain into safe and unsafe regions. In addition, we employ clustering and symbolic representation procedures that compress the unsafe regions into a compact representation. This, in turn, allows to *temporarily* activate the shield solely in (potentially) unsafe regions, in an efficient manner. Our novel approach allows to *significantly* reduce runtime overhead while still preserving formal safety guarantees. We extensively evaluate our approach on two benchmarks from the robotic navigation domain, as well as provide an in-depth analysis of its scalability and completeness.

1 Introduction

Deep reinforcement learning (DRL) is gaining popularity due to its recent success in solving complex decision-making problems across various domains and settings (Mnih et al., 2013; Kober et al., 2013; Rolf et al., 2023; Karamzade et al., 2024). However, upon formal and rigorous analysis, even policies generated by state-of-the-art algorithms exhibit a significant drawback: they can not ensure the correctness of the DRL policy for every possible input (Katz et al., 2019b; Corsi et al., 2021). This limitation hinders the full integration of DRL agents in safety-critical scenarios, such as autonomous navigation systems (Tai et al., 2017), robotic controllers (Aractingi et al., 2023), healthcare (Pore et al., 2021), and decision support in regulated industries (Singh et al., 2022) in which even a single mistake can have dire consequences. This setback emphasizes the need for ensuring the absolute compliance of DRL policies with user-specified safety and behavioral requirements (Ray et al., 2019; Ma et al., 2024). To this end, the DRL community has recently made significant

[*] *Both authors contributed equally.*

efforts to generate more reliable agents. These attempts include online training methods such as constrained optimization (Yang et al., 2022a; Zhang et al., 2020) and safe exploration (Simão et al., 2021; Kamran et al., 2022). However, despite promising results, these approaches are heuristic in nature and are unable to guarantee the absolute correctness of the DRL policy in question. This limitation is observed even when the policy is generated with state-of-the-art algorithms, and for performing relatively simple tasks (Corsi et al., 2024).

An alternative family of approaches tackles the DRL safety problem from a different perspective, by decoupling the safety requirements from the training procedure. These techniques typically involve a formal analysis of the neural network function or the integration of various types of domain expert knowledge into the policy. Two of the most promising approaches in this context are formal verification (Liu et al., 2019) and shielding (Bloem et al., 2015). Unlike training-based methods, these techniques are indeed able to provide rigorous guarantees, but they suffer from various limitations. Formal verification, for example, is computationally hard (Katz et al., 2017), and its applicability to various use cases is thus limited (e.g., it is not clear how to verify large language models). In addition, formal verification tools typically return a binary answer, indicating whether the safety requirement holds or not, without providing any alternative solution when the latter occurs, and the policy is deemed unsafe. On the other hand, shielding techniques introduce an external component (i.e., a "shield"), that can override the original unsafe decisions, hence providing a safe action when encountering an unsafe input. However, although shielding affords safety certifications, there is still no guarantee that the proposed action is optimal (Alshiekh et al., 2018) and, crucially, the external shield must be invoked in every time step, resulting in significant overhead. This issue is critical, because in many real-time applications such an overhead may be infeasible in practice.

In this work, we begin bridging this gap, and present *verification-guided shielding*, a novel method that combines both these aforementioned techniques. Our approach consists of two main stages. First, we employ a combination of different formal methods to identify all the regions in the input space where the agent is guaranteed to behave correctly. In these regions, we can rely on the original policy, without invoking the external shield to validate the agent's decisions (we note that this is possible only due to the rigorous guarantee provided by the formal verification process). Then, in the remaining (unsafe) input region, we activate the shield, which can potentially override the unsafe decision, when encountered. Our approach significantly reduces the overall overhead of "traditional shielding", while still preserving the formal guarantees regarding the policy's safety. Implementing our approach poses several challenges, ranging from scalability to the soundness of the algorithm, which we thoroughly analyze in the following sections. Finally, to demonstrate the effectiveness of our approach, we extensively evaluate it on two popular DRL benchmarks: (i) `Particle World`, where an agent is trained to navigate in a two-dimensional grid, and (ii) `Mapless Navigation`, a real-world task in robotics, where a robot learns to navigate in an unknown arena and reach a given target (Pore et al., 2021; Corsi et al., 2021). We use expressive shields for such tasks (Rodriguez et al., 2024).

The rest of the paper is organized as follows. Sec. 2 contains background on safe DRL, formal verification, and shielding. We formalize our problem in Sec. 3. In Sec. 4, we present our novel method for verification-guided shielding, and empirically evaluate it in Sec. 5. Related work is covered in Sec. 6, and we conclude in Sec. 7.

2 Preliminaries

Deep reinforcement learning algorithms typically aim to optimize the *expected cumulative reward*, which represents the main objective of the agent (Sutton & Barto, 2018). However, in safety-critical tasks, it is common to introduce an additional function that represents the safety constraints that should be met as part of the optimization process. Finding a successful policy under these multiple objectives has emerged as a challenging problem (Ma et al., 2024). Moreover, it is important to note that DRL training algorithms are designed to fulfill requirements only in expectation, without any formal guarantee on the behavior of the policy during deployment.

2.1 Formal Verification

In recent years, various methods have been put forth to formally verify the correctness of deep neural networks (DNNs). These approaches *rigorously* verify whether a given DNN adheres to a safety specification, for *every possible input*. More formally, the DNN verification problem (Katz et al., 2017) is defined as follows:

Definition 1 (*The DNN-Verification Problem*)**.**

Input: $\mathcal{R} = \langle \mathcal{N}, \mathcal{P}, \mathcal{Q} \rangle$, *where* $\mathcal{N}$ *is a DNN,* $\mathcal{P}$ *is a precondition on the DNN's inputs, and* $\mathcal{Q}$ *is a postcondition on the DNN's outputs.*
Output: `SAT` *if* $\exists\, x \mid \mathcal{P}(x) \wedge \mathcal{Q}(\mathcal{N}(x))$*, and* `UNSAT` *otherwise.*

The precondition $\mathcal{P}$ usually encodes domain-specific knowledge on the input space, e.g., it can limit the inputs to represent a specific dangerous situation. The postcondition $\mathcal{Q}$ encodes the *negation* of the desired behavior when the agent's current state belongs to $\mathcal{P}$. Hence, when a verification algorithm (the "verifier") answers `UNSAT`, i.e., that there does not exist a satisfying assignment, this indicates that the DNN behaves correctly on all inputs in our domain of interest. On the other hand, when the verification algorithm returns `SAT`, this indicates that a satisfying assignment is found, and at least a single input x adheres to $\mathcal{P}(x) \wedge \mathcal{Q}(\mathcal{N}(x))$, and triggers the unwanted behavior. The DNN verification problem is computationally hard and has been proven to be NP-complete (Katz et al., 2017), hence, such techniques are usually applied only in safety-critical tasks, in which the safety of the DRL in question must be rigorously guaranteed, and classic testing techniques are inadequate.

DNN Verification Example. DNN verification can be employed in many real-world problems. For instance, it has been shown that DNNs are susceptible to *adversarial inputs*, i.e., small input perturbations that can cause even the best DNNs to fail miserably (Szegedy et al., 2013; Huang et al., 2017a; Ma et al., 2020; Ferhat & Yildirim-Yayilgan, 2020; Gongye et al., 2020). The resilience, or *robustness*, of a DNN to such perturbations can directly be assessed using off-the-shelf verifiers (Tjeng et al., 2017; Zhang et al., 2018; Gopinath et al., 2018; Casadio et al., 2022) by encoding as a precondition an ϵ-ball around a given input x ($\mathcal{P}(x) \coloneqq x \in B_\epsilon(x)$), and as a postcondition a case in which the DNN *misclassifies* a given input $x' \in B_\epsilon(x)$. In the context of deep reinforcement learning, the verified properties are typically safety constraints, that are also encoded (by domain experts) as input-output relations. For additional details, see Appendix D.

2.2 LTL Synthesis and Shielding

Linear temporal logic (LTL) is a type of logic pertaining to modalities referring to linear time (Pnueli, 1977; Manna & Pnueli, 1995). In LTL, it is possible to encode formulae regarding the various states and actions throughout multiple time-steps, e.g., *there are no three consecutive states in which a given action is chosen*. More formally, the LTL syntax is recursively defined as follows:

$$\varphi ::= \top \mid a \mid \varphi \vee \varphi \mid \neg\varphi \mid \bigcirc\varphi \mid \Box\varphi \mid \varphi\, \mathcal{U}\, \varphi,$$

where $a \in \mathsf{AP}$ is an *atomic proposition*, $\{\wedge, \neg\}$ are the common Boolean operators of *conjunction* and *negation*, respectively, and $\{\bigcirc, \mathcal{U}, \Box\}$ are the *next*, *until* and *always* temporal operators, respectively. Reactive LTL synthesis (Piterman et al., 2006; Thomas, 2008) is the task of automatically producing a system that satisfies a given LTL specification φ, where atomic propositions in φ are split into variables assigned by an uncontrollable environment (input variables I) and variables assigned by a controllable system (output variables O). We refer the reader to Appendix B for an in-depth description of LTL semantics and synthesis.

DRL Shielding. Recently, it has been shown that a given LTL formula φ represents a desired specification that can be used to automatically synthesize shields (Bloem et al., 2015; Alshiekh et al., 2018), i.e., generate external components that are coupled with the agent, and *force* it to behave safely according to the specification φ. More formally, given an LTL specification φ,

it is possible to generate a shield S with respect to a given system D (controlled by a DRL agent, in our case). S guarantees that *all* behaviors of the DRL-controlled system D satisfy φ as follows: when S encounters an input I that triggers an erroneous output (i.e., $D(I) := O$ for which $\varphi(I, O)$ does not hold), the original action O is corrected, and replaced with another action O', ensuring that φ(I, O') does hold. This scheme, depicted in Fig. 1, guarantees that the combined system $D \circ S$ never violates φ.

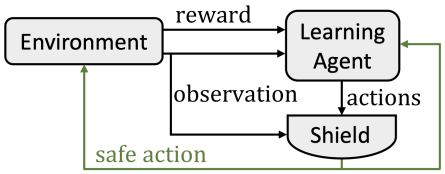

Figure 1: A shielding architecture scheme for a DRL agent (Alshiekh et al., 2018).

Example. Given the atomic proposition `COLLIDE` (indicating that the agent collided), the LTL formula $\phi := \square(\neg\texttt{COLLIDE})$ encodes the safety property in which for all steps ("always"), no collision occurs. Given this requirement, and given that a DRL-controlled agent D observes an input I representing an obstacle to the left — an action $O :=$ `TURN LEFT` will cause a collision in the next step (hence violating ϕ). Thus, a shield S may override O with an alternative action (e.g., $O' :=$ `TURN RIGHT`), satisfying ϕ and hence maintaining safety.

3 Motivation, Benchmarks, and Problem Formulation

Benchmarks. In our evaluation, we focused on two popular DRL benchmarks: (i) `Particle World`, in which an agent moves in a simple two-dimensional grid trying to reach a target position while avoiding collisions with obstacles; and (ii) `Mapless Navigation`, a real-world robotic navigation task, in which a robotic agent navigates in an unknown arena by relying only on local sensors. Both benchmarks are extensively studied in the context of safe DRL (Marchesini et al., 2022; Amir et al., 2023a; Corsi et al., 2024) given their straightforward safety requirements (e.g., collision avoidance). For a more detailed description of the environment and training setup, see Appendix A.

Experimental Setting. We extensively trained more than 250 agents on each of these tasks, with the state-of-the-art PPO algorithm (Schulman et al., 2017) for 500 episodes. All agents shared the same architecture and differed solely in the random seed used to generate their initial parameters. As can be seen in Fig. 2, in both benchmarks, the trained policies reached an average *success rate* (i.e., number of successful trajectories) of over 90%. Next, we selected per each benchmark, the five best-performing models and analyzed their performance from a safety perspective, as summarized in Tab. 1.

Motivation. All five models attained an average success rate of approximately 95%, and also a (seemingly) safe decision-making policy: in 100 randomly generated trials, not a single collision was recorded. However, when analyzed through the lens of formal verification, we identified that *all* the selected models had input configurations in which the policies can indeed behave unsafely and collide with a wall (see the rightmost column of Tab. 1). We believe this further motivates our work —

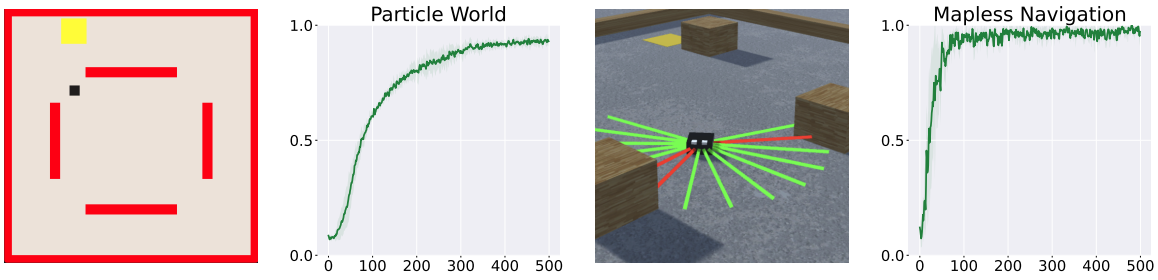

Figure 2: The environments analyzed in our evaluation: `Particle World` (left) and `Mapless Navigation` (right). The plot displays the number of episodes on the x-axis and the corresponding success rate on the y-axis.

Seed	Empirical Success (%)	Empirical Collisions (%)	Verification Output
12	95.6	0.0	`SAT`
66	97.3	0.0	`SAT`
239	91.3	0.0	`SAT`
251	95.3	0.0	`SAT`
258	94.2	0.0	`SAT`

Table 1: Results of the formal analysis for the top five models trained on the `Particle World` environment. A `SAT` verification output indicates the existence of unsafe behaviors.

although the models were extensively trained with a state-of-the-art algorithm to solve a (relatively) simple task, and although they seemed to behave safely in empirical evaluation, this was indeed not the case, as formal methods were able to uncover input configurations in which they fail miserably.

3.1 Model Selection via Verification

The well-known susceptibility of DNNs in general, and DRL agents in particular, to adversarial inputs renders it unlikely to find models that *always* behave safely across the whole input domain, even when trained for relatively simple tasks (Casadio et al., 2022; Amir et al., 2023a; Corsi et al., 2024). This phenomenon has also been confirmed by our evaluation reported in Tab. 1: even when formally analyzing hundreds of trained models with near-perfect empirical performance, not a single model was found to be safe throughout the entire input domain. Moreover, as discussed in Sec. 2, formal verification algorithms can detect (offline) whether a DNN is unsafe or not, however, once a DNN is deemed unsafe, it is not clear what practitioners should do. We believe that both aforementioned limitations further motivate the need for shielding.

3.2 Shielding Policies in Particle Domain

Shield synthesis is the logic-based procedure of generating a shield corresponding to an LTL specification φ, as explained briefly in Sec. 2. This process builds upon encoding Boolean predicates that represent the various input variables, and hence, shield synthesis is usually geared towards a finite state space. However, in most DRL use cases, including the ones covered here, there is an infinite input domain, for example, the continuous input space of `Particle World`. Still, it has very recently been proven that the task of synthesizing shields for such cases (formally known as LTL modulo first-order theories), is decidable (Rodriguez & Sánchez, 2023; 2024b) for various cases pertaining to the temporal logic encodings of φ. Building upon these results, Rodriguez & Sánchez (2024a) presents a novel technique that can be leveraged to synthesize shields in such scenarios (Rodriguez et al., 2024), which we use in this paper. We also note that the shield synthesis procedure can be expedited in various cases. For example, many specifications of interest are in the form $\varphi := \square\varphi'$, where φ' is free of temporal operators. In such cases, the synthesis process can be significantly optimized and computed with alternative runtime enforcement methods (Cassandras & Lafortune, 1999; Falcone et al., 2012). Note that the shield both checks the correctness of the original action and provides the corrected action, when necessary.

Seed	w/o Shield Collisions (%)	w/ Shield Collisions (%)	Interventions (%)	Overhead
12	0.33	0.0	9.6	40.0×
66	0.21	0.0	5.6	32.5×
239	0.27	0.0	5.3	36.3×
251	0.41	0.0	11.0	31.1×
258	0.62	0.0	10.9	35.5×

Table 2: Overhead due to standard shielding; the first two columns demonstrate how the shield can prevent collisions while introducing a significant overhead, even though fewer than 9% of the actions are overridden on average across seeds. Data is collected from the `Particle World` environment.

Tab. 2 summarizes the safety of five trained DRL policies, with and without shielding. Although the shield's soundness indeed guarantees absolute safety (see the third column indicating no collisions), our results demonstrate the main limitations of current shielding approaches: the shield is activated *online* in every time-step throughout the *whole* input domain, resulting in a significant overhead during deployment. From our preliminary analysis, we found that in most cases the shield does not override the original action (i.e., no interventions; see fourth column), demonstrating the safety of the original behavior in the majority of cases. However, even in such situations, the overhead remains due to the constant shield execution. In the following section, we propose an approach that takes advantage of this and drastically reduces the number of calls to the shield while still guaranteeing safety throughout the entire input domain.

4 Verification-Guided Shielding

Next, building on the previous findings, we present our novel *verification-driven shielding* approach. We devised this method using recent advances in formal verification, shielding, and symbolic representation. Our approach incorporates five steps: (1) domain splitting, after which we perform (2) safe-region verification, and (3) clustering. Subsequently, we (4) encode a symbolic representation of the input domain, which finally allows (5) shielding the agent only on potentially unsafe regions.

(1) Domain Splitting. In the first step, our method employs an off-the-shelf verification algorithm to identify all the regions of the state space in which a given DRL agent behaves correctly. This process mainly relies on the concept of *All-DNN-Verification* (Marzari et al., 2023). In essence, this includes pruning the input region in search of all regions in which the trained agent is provably safe, with respect to a set of given requirements. More formally, we search for regions in which the negated (safety) property is `UNSAT`. An exact solution to this problem would provide the complement of the regions where the agent potentially requires a shield.

However, given that this problem is #P-hard (Marzari et al., 2023), the authors proposed ϵ-`ProVe`, an algorithm that computes an underapproximation of these safe regions. In more detail, ϵ-`ProVe` divides the input domain into regions, effectively generating a search tree where each node represents a partition of the input domain; these regions are then analyzed using a sampling approach that provides an estimated probability that the region is safe. The algorithm iteratively splits regions into subregions, until it cannot find any counterexamples (i.e., `UNSAT` assignments to the negated property), in which case the region is approximated as safe. Otherwise, the algorithm heuristically decides whether to declare the entire region as unsafe or continue with the splitting procedure. For a detailed description of ϵ-`ProVe`, and a discussion regarding the probabilistic guarantees provided, we refer the reader to (Marzari et al., 2023).

(2) Formal Verification of Safe Regions. Subsequently, we are left with a division of the input domain into regions, with each region approximated as either safe or unsafe. Although ϵ-`ProVe` typically provides tight results with high confidence, the approximated nature of the approach is not enough to guarantee *absolute* correctness, which is the subject matter of this work. To address this gap, we complement the approximated regions by formally verifying the regions previously approximated as safe. Toward this end, we employ `Marabou`, a sound and complete verification tool (Katz et al., 2019b; Wu et al., 2024), which is used to formally certify the safety of the agent only in the regions that are previously approximated as safe. In this second, fined-tuned verification procedure, if a counterexample (`SAT`) is found in a (mistakenly approximated) safe region, we reclassify the region as unsafe. On the other hand, regions that have already been found to violate the property (ϵ-`ProVe` returned `SAT` in the first step), are left untouched, as a valid counterexample was already found. A pseudocode describing this procedure can be found in Appendix E.

(3) Clustering. After these first two steps, we are left with a sound division of the input space into regions in which the agent is provably safe and regions in which there is at least one input configuration that causes the agent to behave unsafely. Next, we would like to apply our synthesized

shield solely on these potentially unsafe regions. However, this presents a new challenge as the set $\mathcal{S}$ of unsafe regions includes, in practice, a large number of compact regions (e.g., we observed an average cardinality of $\approx 60,000$ in `Particle World`). While this does not compromise the correctness of our strategy, it raises another problem — checking whether the current input belongs to the set $\mathcal{S}$ introduces significant overhead, potentially mitigating the benefits of our approach. To address this limitation, we employ an additional step in which we cluster the set of unsafe regions and reduce their overall cardinality. Specifically, we employ *agglomerative clustering* (Ackermann et al., 2014), to concatenate unsafe regions and produce an overapproximation of the unsafe regions. As we later demonstrate, this *significantly* reduces the number of unsafe regions and, consequently, the overhead for checking whether the current state belongs to $\mathcal{S}$. It is important to emphasize that, although the clustering step has the potential to overapproximate safe regions as unsafe (see Fig. 3), it does not compromise the overall soundness of our approach. At most, the shield may be activated more than strictly necessary.

(4) Symbolic Representation. Next, we make use of symbolic representation (Hoffmann et al., 2007), i.e., an encoding of all the states in order to obtain a succinct formula for all unsafe regions. Due to our focus on fully observable input domains, we can use propositional logic formulas to symbolically encode the regions of interest (in our case, unsafe regions). Furthermore, this formula can be reduced to a succinct equivalent formula, e.g., by using off-the-shelf solvers (De Moura & Bjørner, 2008; Barrett & Tinelli, 2018). This, in turn, could potentially reduce the overhead of checking online whether the agent is in an unsafe region. We elaborate further on symbolic representation in Appendix C.

(5) Shield Synthesis and Execution. Finally, we are left with a set of (relatively) few approximated unsafe regions. First, we can synthesize a shield that, when activated, guarantees safety with respect to the safety property ψ. Next, we can couple the shield with the agent, and at each time-step: (i) efficiently identify if the current input belongs to the potentially unsafe regions, and if so, (ii) temporarily activate the synthesized shield and guarantee the safety, as previously described in Sec. 3. In the remaining (provably) safe regions, the shield can remain inactive while preserving the formal guarantees, as we already verified that any original decision that the agent makes is safe.

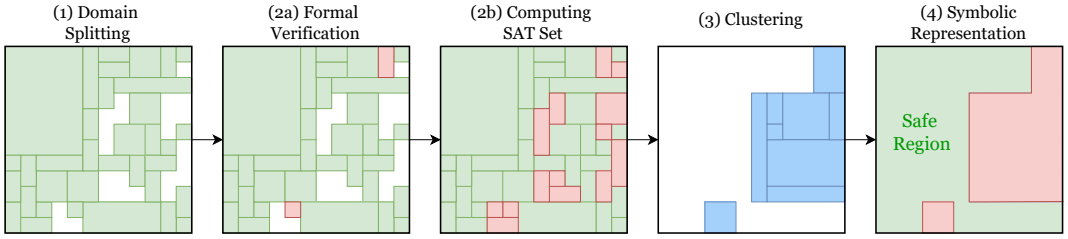

Figure 3: An overview of *verification-guided shielding*. In step (1) we employ ϵ-`ProVe` to split the input domain into approximated safe (green) and unsafe regions; (2a) these can be further validated with a formal verification tool, (2b) which complements the set of approximated unsafe regions with the ones formally found as such. (3) To reduce the cardinality of this set we employ a clustering algorithm and (4) further simplify the encoded results by using symbolic representation.

5 Empirical Evaluation

Our experimental evaluations comprise two components: the *offline* procedures for generating the shield and identifying safe regions, and the *online* execution of the system, where the goal is to minimize the computational overhead resulting from invoking the shield.

Experimental Setup. The offline evaluation was conducted on a distributed cluster with 160 CPUs and 448GB RAM. For each verification query, we employ 1 CPU, 1GB RAM, and a runtime

Seed	Splitting (s)	Verification (hr)	Clustering (s)	Synthesis (s)	Reduction (s)
12	251.7	1.44	12.28		22.48
66	239.1	2.01	14.02		29.12
239	458.0	2.17	120.84	2.69	38.67
251	451.2	2.26	139.30		38.93
258	485.4	2.23	248.55		43.62

Table 3: `Particle World`: the overall time required for the offline components of our approach.

limit of three hours. We also note that in the case of the verification queries, Marabou internally used the *Guorobi* LP solver (Anand et al., 2017) as a backend engine. Data related to the overhead was collected on a commercial laptop to align with the limited hardware typically used to operate autonomous robotic systems.

Offline Procedures. Tab. 3 summarizes the time measured for each of the offline steps with respect to the `Particle World` benchmark. The most time-consuming component is the formal verification (step 2), taking an average of over two hours; this is not surprising, as in this step the verifier is required to solve many NP-complete problems, per each policy. However, we believe this is a reasonable price as (i) this is an offline step that is executed once; and (ii) this is the step that provides the formal guarantees. On the other hand, ϵ-`ProVe` (i.e., the splitting procedure in step 1) runs significantly faster but provides only probabilistic assurances. Both stages are complementary and represent a simulated annealing-like optimization (Kirkpatrick et al., 1983): at first, we approximate and reduce the number of regions on which we can rule out correct behavior, and then, we run a more expensive, formal verification procedure that fine-tunes the remaining regions. The column representing the clustering demonstrates a high variance among the different policies, as there can be significant differences in the number of unsafe regions identified in the previous steps. Still, it is worth noting that even in the worst-case measurement, the time required by this procedure is negligible when compared to the formal verification step. Finally, the table reports the results of the shield synthesis and the formula reduction steps, which were not particularly time-demanding (i.e., step 4). These results also align with the hypothesis raised in the previous sections, i.e., that the heavy computational cost of shielding is not related to the offline synthesis time, but rather to the cost of invoking the shield online, before each decision.

Online Invocation. Our main results are presented in Tab. 4. Specifically, we compare the overhead introduced by the shield in two cases: the classic fully-activated shielding approach and our verification-guided approach. The first half of the table reports the analysis on the `Particle World` environment, while the second one reports the results for `Mapless Navigation`. In general, both benchmarks demonstrate the merits of our approach in reducing significant overhead, confirming

Seed	**Full Shield**		**Verification-Guided Shield**		**Gain (%)**
	Active Time (%)	Overhead	Active Time (%)	Overhead	
12	100	40.0×	28.6	14.1×	64.8
66	100	32.5×	32.4	13.1×	59.7
239	100	36.3×	44.5	21.5×	40.7
251	100	31.1×	37.6	13.2×	57.6
258	100	35.5×	33.8	13.9×	60.1
104	100	4.8×	61.7	3.6×	25.1
225	100	4.4×	53.1	3.5×	20.5
239	100	4.5×	2.1	1.8×	60.0
243	100	4.5×	1.3	1.6×	71.1
310	100	4.6×	3.4	1.5×	67.4

Table 4: Final results; the first block presents the results for the `Particle World` benchmark and the second block represents the results for the `Mapless Navigation` benchmark.

the general environment nature of our methodology. In more detail, per each environment and seed, we compare the portion of time in which the shield was activated during execution. This value is computed by normalizing the number of shield invocations by the total number of actions. While in the first column, the (full) shield is trivially always active, we observe a drastic reduction when our approach has been applied, especially in some `Mapless Navigation` seeds. We also note that there is not necessarily a direct correlation between the size of the unsafe regions and the number of interventions, as the interventions were measured with respect to stochastic executions. Finally, we report the average computational time overhead as a relative value compared to an episode with the shield deactivated, i.e., decisions made without invoking any external components. Not surprisingly, our results show a clear correlation between the *active time* and the overhead introduced, further motivating this work. In the last column, we summarize the time gain provided by our method with respect to invoking the shield at every time-step. Note that the gain is not always proportional to the *active time*; the resulting value also depends on the actual number of steps required to complete a single episode, i.e., the absolute number of calls to the shield.

Note. Per each benchmark, we encoded $1,000$ LTL formulas for our shield. We emphasize that this number does not affect the relative gain of our *verification-guided shield*, but only affects the *absolute* value of the overhead compared to a single forward propagation. We also emphasize that the formal verification was skipped in `Mapless Navigation`, in order to expedite the procedure, and hence we relied only on the probabilistic guarantees afforded by ϵ-`ProVe`. In addition, we note that our evaluation included the first three steps, while excluding the fourth step, i.e., symbolic representation, as this step ran slightly slower than when using the complete set of input regions. Still, we executed this step and reported the overall time that it took, while successfully demonstrating that symbolic representations can be encoded for environments including thousands of states. Since the efficiency of this step highly depends on the task in question and the underlying SMT solver, improving symbolic representations is beyond the scope of this work.

Limitations. Although these results are encouraging, it is important to acknowledge certain limitations of our approach, mainly inherited from the backend shielding and verification techniques. First, our approach requires a valid encoding of the required properties of interest. This, in turn, assumes access to the environment dynamics and the agent's transition model, as well as the practitioner's ability to encode the relevant properties in a logic-based form. In addition, when relying on shielding in unsafe regions, it is important to note that although the shield guarantees adherence to the given requirements, it does not necessarily select the *optimal* action, among the safe ones. Furthermore, there are some limitations due to the backend DNN verification tools. Specifically, our approach relies on `Marabou`, which affords only limited support to some activation functions, hence restricting its applicability to some advanced DNN architectures. However, we believe that these limitations can serve as a foundation for future research.

6 Related Work

In recent years, the formal methods community has put forth a wide range of approaches and tools for formally verifying the correctness of deep learning models (Tjeng et al., 2017; Lomuscio & Maganti, 2017; Huang et al., 2017b; Wang et al., 2018; Gehr et al., 2018; Kuper et al., 2018; Gopinath et al., 2018; Singh et al., 2019; Lyu et al., 2020; Katz et al., 2021; Wu et al., 2024). In addition, there has recently been ample research on formally verifying DRL systems (Fulton & Platzer, 2018; Dutta et al., 2018; 2019; Sun et al., 2019; Vasić et al., 2022; Mandal et al., 2024), in particular in the context of safety (Kazak et al., 2019; Amir et al., 2021a; Eliyahu et al., 2021) and explainability (Bassan & Katz, 2023; Bassan et al., 2023). Other work focused on enhancing DRL safety by inducing Scenario-Based Programming (SBP) (Corsi et al., 2022; Yerushalmi et al., 2022; 2023).

Classic shielding approaches (Alshiekh et al., 2018; Pranger et al., 2021a;b) focus on properties expressed in Boolean LTL and are incompatible with systems pertaining to richer data domains. However, more recently, Wu et al. (2019) presented the concept of (incomplete) shields for linear arithmetic. To address these limitations, the work of Rodriguez & Sánchez (2023; 2024a) proves

that, under certain conditions, LTL synthesis with data specifications is decidable via abstraction methods, which can be used for reactive synthesis of expressive shields with e.g., numerical information (Rodriguez et al., 2024). Additional techniques, such as runtime enforcement and supervisory control (Cassandras & Lafortune, 1999; Schneider, 2000; Ligatti et al., 2009; Falcone et al., 2012), share similarities with shielding, however, such methods are incompatible with DRL and general reactive systems, but rather, focus solely on checking software invariants.

We also note that in addition to formal verification and shielding, there exist other popular approaches for improving the safety of DRL agents. These methods are typically applied during training and rely on constrained optimization (Stooke et al., 2020; Liu et al., 2020; Roy et al., 2021), safe exploration (Srinivasan et al., 2020; Yang et al., 2022b), and various alternative solutions (García & Fernández, 2015; Achiam et al., 2017; Tessler et al., 2019). However, although popular, these techniques are heuristic in nature and do not afford any formal guarantees regarding the safety of the DRL agent in question (Brunke et al., 2021).

7 Conclusion

In this paper, we combine verification and shielding and propose a novel technique that leverages the advantages of both these formal approaches. Specifically, we demonstrate how to use formal methods to prune the input space and divide it into safe and (potentially) unsafe regions. While in the first case, we can safely employ the original, shield-less model, in the latter we activate the shield online, and override any potential unsafe action. We extensively evaluate our approach in multiple experiments, and demonstrate that it drastically reduces the overhead of shielding, while maintaining guaranteed safety throughout the whole input domain.

Moving forward, our approach can be extended along various axes. Currently, we employ clustering algorithms to approximate unsafe regions, however, we plan to investigate additional strategies that attain more compact descriptions and reduce the use of shielding even further. Additionally, we plan to incorporate our approach also into the DRL training phase to iteratively robustify the agents prior to deployment. Finally, we plan to explore alternative strategies, such as deep ensembles, to further reduce the need for shield interventions. We believe this work is another step towards the reliable use of DRL in safety-critical domains.

Acknowledgments

The work of Corsi and Fox was funded in part by the National Science Foundation (Award #2321786). The work of Amir and Katz was partially funded by the European Union (ERC, VeriDeL, 101112713). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. The work of Amir was further supported by a scholarship from the Clore Israel Foundation. The work of Rodríguez and Sánchez was funded in part by PRODIGY Project (TED2021-132464B-I00) — funded by MCIN/AEI/10.13039/501100011033/ and the European Union NextGenerationEU/PRTR — by the DECO Project (PID2022-138072OB-I00) — funded by MCIN/AEI/10.13039/501100011033 and by the ESF, as well as by a research grant from Nomadic Labs and the Tezos Foundation.

References

J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained Policy Optimization. In *Int. Conf. on Machine Learning*, pp. 22–31. PMLR, 2017.

M. Ackermann, J. Blömer, D. Kuntze, and C. Sohler. Analysis of Agglomerative Clustering. *Algorithmica*, 2014.

M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. Safe Reinforcement Learning via Shielding. In *Proc. of the 32nd AAAI Conference on Artificial Intelligence*, pp. 2669–2678, 2018.

G. Amir, M. Schapira, and G. Katz. Towards Scalable Verification of Deep Reinforcement Learning. In *Proc. 21st Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pp. 193–203, 2021a.

G. Amir, H. Wu, C. Barrett, and G. Katz. An SMT-Based Approach for Verifying Binarized Neural Networks. In *Proc. 27th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 203–222, 2021b.

G. Amir, T. Zelazny, G. Katz, and M. Schapira. Verification-Aided Deep Ensemble Selection. In *Proc. 22nd Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pp. 27–37, 2022.

G. Amir, D. Corsi, R. Yerushalmi, L. Marzari, D. Harel, A. Farinelli, and G. Katz. Verifying Learning-Based Robotic Navigation Systems. In *Proc. 29th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 607–627, 2023a.

G. Amir, Z. Freund, G. Katz, E. Mandelbaum, and I. Refaeli. veriFIRE: Verifying an Industrial, Learning-Based Wildfire Detection System. In *Proc. 25th Int. Symposium on Formal Methods (FM)*, pp. 648–656, 2023b.

G. Amir, O. Maayan, T. Zelazny, G. Katz, and M. Schapira. Verifying Generalization in Deep Learning. In *Proc. 35th Int. Conf. on Computer Aided Verification (CAV)*, pp. 438–455, 2023c.

R. Anand, D. Aggarwal, and V. Kumar. A Comparative Analysis of Optimization Solvers. *Journal of Statistics and Management Systems*, 20(4):623–635, 2017.

M. Aractingi, P. Léziart, T. Flayols, J. Perez, T. Silander, and P. Souères. Controlling the Solo12 Quadruped Robot with Deep Reinforcement Learning, 2023.

C. Barrett and C. Tinelli. *Satisfiability Modulo Theories*. Springer, 2018.

S. Bassan and G. Katz. Towards Formal Approximated Minimal Explanations of Neural Networks. In *Proc. 29th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 187–207, 2023.

S. Bassan, G. Amir, D. Corsi, I. Refaeli, and G. Katz. Formally Explaining Neural Networks within Reactive Systems. In *Proc. 23rd Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pp. 10–22, 2023.

R. Bloem, B. Könighofer, R. Könighofer, and C. Wang. Shield Synthesis: - Runtime Enforcement for Reactive Systems. In *Proc. of the 21st Int. Conf. in Tools and Algorithms for the Construction and Analysis of Systems, (TACAS)*, volume 9035, pp. 533–548, 2015.

L. Brunke, M. Greeff, A. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. Schoellig. Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 5, 2021.

M. Casadio, E. Komendantskaya, M. Daggitt, W. Kokke, G. Katz, G. Amir, and I. Refaeli. Neural Network Robustness as a Verification Property: A Principled Case Study. In *Proc. 34th Int. Conf. on Computer Aided Verification (CAV)*, pp. 219–231, 2022.

C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*, volume 11 of *The Kluwer International Series on Discrete Event Dynamic Systems*. Springer, 1999.

D. Corsi, E. Marchesini, and A. Farinelli. Formal Verification of Neural Networks for Safety-Critical Tasks in Deep Reinforcement Learning. In *Proc. 37th Conf. on Uncertainty in Artificial Intelligence (UAI)*, 2021.

D. Corsi, R. Yerushalmi, G. Amir, A. Farinelli, D. Harel, and G. Katz. Constrained Reinforcement Learning for Robotics via Scenario-Based Programming, 2022. Technical Report. https://arxiv.org/abs/2206.09603.

D. Corsi, G. Amir, G. Katz, and A. Farinelli. Analyzing Adversarial Inputs in Deep Reinforcement Learning, 2024. Technical Report. https://arxiv.org/abs/2402.05284.

L. De Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *Proc. 14th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 337–340, 2008.

S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari. Learning and Verification of Feedback Control Systems using Feedforward Neural Networks. *IFAC-PapersOnLine*, 51(16):151–156, 2018.

S. Dutta, X. Chen, and S. Sankaranarayanan. Reachability Analysis for Neural Feedback Systems using Regressive Polynomial Rule Inference. In *Proc. 22nd ACM Int. Conf. on Hybrid Systems: Computation and Control (HSCC)*, pp. 157–168, 2019.

Y. Elboher, J. Gottschlich, and G. Katz. An Abstraction-Based Framework for Neural Network Verification. In *Proc. 32nd Int. Conf. on Computer Aided Verification (CAV)*, pp. 43–65, 2020.

Y. Elboher, E. Cohen, and G Katz. Neural Network Verification using Residual Reasoning. In *Proc. 20th Int. Conf. on Software Engineering and Formal Methods (SEFM)*, pp. 173–189, 2022.

T. Eliyahu, Y. Kazak, G. Katz, and M. Schapira. Verifying Learning-Augmented Systems. In *Proc. Conf. of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pp. 305–318, 2021.

Y. Falcone, J. Fernandez, and L. Mounier. What can you Verify and Enforce at Runtime? *Int. Journal on Software Tools for Technology Transfer*, pp. 349–382, 2012.

O. Ferhat and S. Yildirim-Yayilgan. Deep Neural Network Based Malicious Network Activity Detection Under Adversarial Machine Learning Attacks. In *Proc. 3rd Int. Conf. on Intelligent Technologies and Applications (INTAP)*, pp. 280–291, 2020.

N. Fulton and A. Platzer. Safe Reinforcement Learning via Formal Methods: Toward Safe Control through Proof and Learning. In *Proc. 32nd AAAI Conf. on Artificial Intelligence (AAAI)*, 2018.

J. Garcıa and F. Fernández. A Comprehensive Survey on Safe Reinforcement Learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

T. Gehr, M. Mirman, D. Drachsler-Cohen, E. Tsankov, S. Chaudhuri, and M. Vechev. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *Proc. 39th IEEE Symposium on Security and Privacy (S&P)*, 2018.

C. Gongye, H. Li, X. Zhang, M. Sabbagh, G. Yuan, X. Lin, T. Wahl, and Y. Fei. New Passive and Active Attacks on Deep Neural Networks in Medical Applications. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pp. 1–9, 2020.

D. Gopinath, G. Katz, C. Păsăreanu, and C. Barrett. DeepSafe: A Data-driven Approach for Assessing Robustness of Neural Networks. In *Proc. 16th. Int. Symposium on Automated Technology for Verification and Analysis (ATVA)*, pp. 3–19, 2018.

J. Hoffmann, C. Gomes, B. Selman, and H. Kautz. SAT Encodings of State-Space Reachability Problems in Numeric Domains. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 1918–1923, 2007.

S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel. Adversarial Attacks on Neural Network Policies, 2017a. Technical Report. https://arxiv.org/abs/1702.02284.

X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety Verification of Deep Neural Networks. In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, pp. 3–29, 2017b.

D. Kamran, T. Simão, Q. Yang, C. T Ponnambalam, J. Fischer, M. Spaan, and M. Lauer. A Modern Perspective on Safe Automated Driving for Different Traffic Dynamics Using Constrained Reinforcement Learning. In *2022 IEEE 25th Int. Conf. on Intelligent Transportation Systems (ITSC)*, pp. 4017–4023. IEEE, 2022.

A. Karamzade, K. Kim, M. Kalsi, and R. Fox. Reinforcement Learning from Delayed Observations via World Models. *arXiv preprint arXiv:2403.12309*, 2024.

G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, pp. 97–117, 2017.

G. Katz, D. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, D. Dill, M. Kochenderfer, and C. Barrett. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *Proc. 31st Int. Conf. on Computer Aided Verification (CAV)*, pp. 443–452, 2019a.

G. Katz, D. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, D. Dill, M. Kochenderfer, and C. Barrett. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *Proc. 31st Int. Conf. on Computer Aided Verification (CAV)*, pp. 443–452, 2019b.

G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: a Calculus for Reasoning about Deep Neural Networks. *Formal Methods in System Design (FMSD)*, 2021.

Y. Kazak, C. Barrett, G. Katz, and M. Schapira. Verifying Deep-RL-Driven Systems. In *Proc. 1st ACM SIGCOMM Workshop on Network Meets AI & ML (NetAI)*, pp. 83–89, 2019.

S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598): 671–680, 1983.

J. Kober, J. Bagnell, and J. Peters. Reinforcement Llearning in Robotics: A Survey, 2013.

L. Kuper, G. Katz, J. Gottschlich, K. Julian, C. Barrett, and M. Kochenderfer. Toward Scalable Verification for Safety-Critical Deep Networks, 2018. Technical Report. https://arxiv.org/abs/1801.05950.

J. Ligatti, L. Bauer, and D. Walker. Run-Time Enforcement of Nonsafety Policies. *ACM Trans. Inf. Syst. Secur.*, 12(3), 2009.

C. Liu, T. Arnon, C. Lazarus, C. Barrett, and M. Kochenderfer. Algorithms for Verifying Deep Neural Networks, 2019. Technical Report. http://arxiv.org/abs/1903.06758.

Y. Liu, J. Ding, and X. Liu. IPO: Interior-Point Policy Optimization under Constraints. In *Proc. 34th AAAI Conf. on Artificial Intelligence (AAAI)*, pp. 4940–4947, 2020.

A. Lomuscio and L. Maganti. An Approach to Reachability Analysis for Feed-Forward ReLU Neural Networks, 2017. Technical Report. http://arxiv.org/abs/1706.07351.

Z. Lyu, C. Y. Ko, Z. Kong, N. Wong, D. Lin, and L. Daniel. Fastened Crown: Tightened Neural Network Robustness Certificates. In *Proc. 34th AAAI Conf. on Artificial Intelligence (AAAI)*, pp. 5037–5044, 2020.

H. Ma, C. Liu, S. Li, S. Zheng, W. Sun, and J. Chen. Learn Zero-Constraint-Violation Safe Policy in Model-Free Constrained Reinforcement Learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

J. Ma, S. Ding, and Q. Mei. Towards More Practical Adversarial Attacks on Graph Neural Networks. In *Proc. 34th Conf. on Neural Information Processing Systems (NeurIPS)*, 2020.

U. Mandal, G. Amir, H. Wu, I. Daukantas, F. Newell, U. Ravaioli, B. Meng, M. Durling, M. Ganai, T. Shim, G. Katz, and C. Barrett. Formally Verifying Deep Reinforcement Learning Controllers with Lyapunov Barrier Certificates, 2024. Technical Report. https://arxiv.org/abs/2405.14058.

Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer Science & Business Media, 1995.

E. Marchesini and A. Farinelli. Discrete Deep Reinforcement Learning for Mapless Navigation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

E. Marchesini, D. Corsi, and A. Farinelli. Exploring Safer Behaviors for Deep Reinforcement Learning. In *Proc. of the AAAI Conference on Artificial Intelligence*, 2022.

L. Marzari, D. Corsi, E. Marchesini, A. Farinelli, and F. Cicalese. Enumerating Safe Regions in Deep Neural Networks with Provable Probabilistic Guarantees. *arXiv preprint arXiv:2308.09842*, 2023.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning, 2013. Technical Report. https://arxiv.org/abs/1312.5602.

David Monniaux. A Quantifier Elimination Algorithm for Linear Real Arithmetic. In *Proc. of the 15th International Conference in Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2008)*, volume 5330 of *LNCS*, pp. 243–257. Springer, 2008. doi: 10.1007/978-3-540-89439-1_18. URL https://doi.org/10.1007/978-3-540-89439-1_18.

N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of Reactive (1) Designs. In *Proc. 7th Int. Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, pp. 364–380, 2006.

A. Pnueli. The Temporal Logic of Programs. In *Proc. of 18th Annual Symposium on Foundations of Computer Science (SFCS)*, pp. 46–57, 1977.

A. Pore, D. Corsi, E. Marchesini, D. Dall'Alba, A. Casals, A. Farinelli, and P. Fiorini. Safe Reinforcement Learning Using Formal Verification for Tissue Retraction in Autonomous Robotic-Assisted Surgery. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

S. Pranger, B. Könighofer, L. Posch, and R. Bloem. TEMPEST - Synthesis Tool for Reactive Systems and Shields in Probabilistic Environments. In *Proc. 19th Int. Symposium in Automated Technology for Verification and Analysis, (ATVA)*, volume 12971, pp. 222–228, 2021a.

S. Pranger, B. Könighofer, M. Tappler, M. Deixelberger, N. Jansen, and R. Bloem. Adaptive Shielding under Uncertainty. In *American Control Conference, (ACC)*, pp. 3467–3474, 2021b.

A. Ray, J. Achiam, and D. Amodei. Benchmarking Safe Exploration in Deep Reinforcement Learning, 2019.

I. Refaeli and G. Katz. Minimal Multi-Layer Modifications of Deep Neural Networks. In *Proc. 5th Workshop on Formal Methods for ML-Enabled Autonomous Systems (FoMLAS)*, 2022.

A. Rodriguez and C. Sánchez. Boolean Abstractions for Realizabilty Modulo Theories. In *Proc. of the 35th Int. Conf. on Computer Aided Verification (CAV'23)*, 2023.

A. Rodriguez and C. Sánchez. Adaptive Reactive Synthesis for LTL and LTLf Modulo Theories. In *Proc. of the 38th AAAI Conf. on Artificial Intelligence*, pp. 5037–5044, 2024a.

A. Rodriguez and C. Sánchez. Realizability modulo theories. *Journal of Logical and Algebraic Methods in Programming*, pp. 100971, 2024b. ISSN 2352-2208. doi: https://doi.org/10.1016/j.jlamp.2024.100971.

A. Rodriguez, G. Amir, D. Corsi, C. Sanchez, and G. Katz. Shield Synthesis for LTL Modulo Theories , 2024. Technical Report. https://arxiv.org/abs/2406.04184.

B. Rolf, I. Jackson, M. Müller, S. Lang, T. Reggelin, and D. Ivanov. A Review on Reinforcement Learning Algorithms and Applications in Supply Chain Management, 2023.

J. Roy, R. Girgis, J. Romoff, P. Bacon, and C. Pal. Direct Behavior Specification via Constrained Reinforcement Learning, 2021. Technical Report. https://arxiv.org/abs/2112.12228.

F. Schneider. Enforceable Security Policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms, 2017. Technical Report. http://arxiv.org/abs/1707.06347.

T. Simão, N. Jansen, and M. Spaan. AlwaysSafe: Reinforcement Learning without Safety Constraint Violations during Training. In *Proc. of the 20th Int. Conf.e on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2021.

G. Singh, T. Gehr, M. Puschel, and M. Vechev. An Abstract Domain for Certifying Neural Networks. In *Proc. 46th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, 2019.

V. Singh, S. Chen, M. Singhania, B. Nanavati, A. Gupta, et al. How are reinforcement learning and deep learning algorithms used for big data based decision making in financial industries–a review and research agenda. *International Journal of Information Management Data Insights*, 2022.

K. Srinivasan, B. Eysenbach, S. Ha, J. Tan, and C. Finn. Learning to be Safe: Deep RL with a Safety Critic, 2020. Technical Report. https://arxiv.org/abs/2010.14603.

A. Stooke, J. Achiam, and P. Abbeel. Responsive Safety in Reinforcement Learning by Pid Lagrangian Methods. In *Proc. 37th Int. Conf. on Machine Learning (ICML)*, pp. 9133–9143, 2020.

X. Sun, H. Khedr, and Y. Shoukry. Formal Verification of Neural Network Controlled Autonomous Systems. In *Proc. 22nd ACM Int. Conf. on Hybrid Systems: Computation and Control (HSCC)*, 2019.

R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing Properties of Neural Networks, 2013. Technical Report. http://arxiv.org/abs/1312.6199.

L. Tai, G. Paolo, and M. Liu. Virtual-to-Real Deep Reinforcement Learning: Continuous Control of Mobile Robots for Mapless Navigation. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2017.

C. Tessler, D. Mankowitz, and S. Mannor. Reward Constrained Policy Optimization. In *Proc. 7th Int. Conf. on Learning Representations (ICLR)*, 2019.

W. Thomas. Church's Problem and a Tour Through Automata Theory. In *Pillars of Computer Science*, pp. 635–655. Springer, 2008.

V. Tjeng, K. Xiao, and R. Tedrake. Evaluating Robustness of Neural Networks with Mixed Integer Programming, 2017. Technical Report. http://arxiv.org/abs/1711.07356.

M. Vasić, A. Petrović, K. Wang, M. Nikolić, R. Singh, and S. Khurshid. MoËT: Mixture of Expert Trees and its Application to Verifiable Reinforcement Learning. *Neural Networks*, 151:34–47, 2022.

S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Formal Security Analysis of Neural Networks using Symbolic Intervals. In *Proc. 27th USENIX Security Symposium*, pp. 1599–1614, 2018.

T. Wolfgang. Automata on Infinite Objects. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pp. 133–191. Elsevier and MIT Press, 1990.

H. Wu, O. Isac, A. Zeljić, T. Tagomori, M. Daggitt, W. Kokke, I. Refaeli, G. Amir, K. Julian, S. Bassan, P. Huang, O. Lahav, M. Wu, M. Zhang, E. Komendantskaya, G. Katz, and C. Barrett. Marabou 2.0: A Versatile Formal Analyzer of Neural Networks, 2024.

M. Wu, J. Wang, J. Deshmukh, and C. Wang. Shield Synthesis for Real: Enforcing Safety in Cyber-Physical Systems. In *Proc. 19th Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pp. 129–137, 2019.

L. Yang, J. Ji, J. Dai, L. Zhang, B. Zhou, P. Li, Y. Yang, and G. Pan. Constrained Update Projection Approach to Safe Policy Optimization. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022a.

Q. Yang, T. Simão, N. Jansen, S. Tindemans, and M. Spaan. Training and Transferring Safe Policies in Reinforcement Learning. In *AAMAS 2022 Workshop on Adaptive Learning Agents*, 2022b.

R. Yerushalmi, G. Amir, A. Elyasaf, D. Harel, G. Katz, and A. Marron. Scenario-Assisted Deep Reinforcement Learning. In *Proc. 10th Int. Conf. on Model-Driven Engineering and Software Development (MODELSWARD)*, pp. 310–319, 2022.

R. Yerushalmi, G. Amir, A. Elyasaf, D. Harel, G. Katz, and A. Marron. Enhancing Deep Reinforcement Learning with Scenario-Based Modeling. *SN Computer Science*, 4(2):156, 2023.

H. Zhang, T. Weng, P. Chen, C. Hsieh, and L. Daniel. Efficient Neural Network Robustness Certification with General Activation Functions, 2018.

Y. Zhang, Q. Vuong, and K. Ross. First Order Constrained Optimization in Policy Space. *Proc. 34th Conf. on Neural Information Processing Systems (NeurIPS)*, 2020.

A Training Details

This appendix provides additional information regarding the described benchmarks, training setups, and safety requirements. Both analyzed benchmarks are navigation tasks with varying degrees of complexity and realism.

A.1 `Particle World`

Our first environment is depicted in Fig. 2 (left). The goal of the agent (black square) is to reach the target position (yellow square) while avoiding collisions with walls and obstacles; the map is randomly selected from five different configurations. The agent does not have access to the full map of the environment but only to local information, and this makes this task an abstraction of the well-studied robotic *mapless navigation* task (Tai et al., 2017; Corsi et al., 2024). The action and observation spaces are continuous and hence, the agent could move to any possible position in the arena; the agent is equipped with four proximity sensors that detect the distance from the closest obstacle in the respective directions, i.e., `Left, Right, Up, Down`. Although the action space is continuous, at each step, the agent is allowed to move only in one of the aforementioned directions, e.g., *at time-step* t_0 *the agent performs a translation on the left of* 0.321 *units*.

State/Action Spaces and DNN Topology. The structure of the neural network is inspired by recent work in the literature demonstrating that this task can be learned by a simple *multi-layer perceptron* (MLP) encompassing relatively few nodes and hidden layers (Marchesini & Farinelli, 2020). Next, we present a more detailed description of the MLP's structure:

- The *input layer* constitutes 8 neurons: the first 4 neurons represent the distance from the closest obstacle in each direction, the following 2 neurons encode the current position of the agent (x and y coordinates), while the last 2 neurons encode the target's position. All these values are normalized in the interval $[0, 1]$ and can take on any continuous values within this interval.
- Two fully-connected *hidden layers* of 16 neurons each, with *ReLU* activation functions.
- An *output layer* of 4 neurons, each representing the translation action in one of the possible directions (i.e., `Left, Right, Up, Down`); the values are continuous as the agent can translate any distance in a given direction. Crucially, the agent is constrained to move in only one direction at each time-step, hence, we always select the action with the highest value among the four options.

Training. We trained our agents with the state-of-the-art *Proximal Policy Optimization* (PPO) algorithm (Schulman et al., 2017), which is widely considered the state-of-the-art. For this first task we employed a discrete reward function, that provides a positive reward when reaching the target and a negative reward for each collision; formally:

$$R_t = \begin{cases} +5 & \text{target reached} \\ -1 & \text{collision with obstacle} \end{cases}$$

where both conditions represent a terminal state. Following is a list of hyperparameters employed during training:

- *training episodes*: 500
- *number of hidden layers*: 2
- *size of hidden layers*: 16
- *parallel environments*: 1
- *gamma* (γ): 0.995

- *learning rate*: 0.0013
- *memory limit*: None
- *update frequency*: 4096 steps
- *trajectory reduction strategy*: sum
- *epochs*: 50
- *batch number*: 64
- *critic network size*: 2x256
- *PPO clip*: 0.2
- *GAE lambda*: 0.99
- *target kl-divergence*: 0.02
- *max gradient normal*: 0.5

Safety Requirements. As mentioned, `Particle World` is an abstraction of a real-world navigation problem; therefore, the crucial safety requirement is collision avoidance. Given the state and action space of the benchmark, the safety requirements involve only the first 4 inputs (pertaining to the presence of obstacles) and the selected action. For more details regarding the encoding of the verification queries, see Appendix D. From a high-level perspective, the safety requirements can be formalized as follows: *"for any possible combination of agent and target position, the agent must not move towards an obstacle with a step-size larger than the distance to the closest obstacle in that direction"*.

A.2 `Mapless Navigation`

Our second environment is depicted in Fig. 2 (right). Mapless navigation is a popular and well-studied task in the DRL literature (Tai et al., 2017; Ray et al., 2019; Marchesini & Farinelli, 2020). This task is considered quite difficult due to the agent solely relying on local observations. For our experiments, we follow the same configuration presented in previous work in the field (Pore et al., 2021; Amir et al., 2023a). In particular, the agent is equipped with a lidar sensor for obstacle detection, and with GPS and compass inputs for localization. A significant difference between `Mapless Navigation` and `Particle World` is the degree of freedom for the agent. Specifically, in `Mapless Navigation`, the agent can simultaneously perform a linear step and a rotation, which provides additional movement options.

State/Action Spaces and DNN Topology. The structure of the neural network is similar to the one we employed for `Particle World`, with the additional features derived from the sensors and actuators (Ray et al., 2019). Following is a more detailed description of the structure:

- The *input layer* constitutes of 9 neurons, the first 7 neurons represent lidar sensor readings, that indicate the distance from an obstacle in a given direction (from left to right, with a step of 30°). The final two input neurons indicate the target's position relative to the agent (i.e., polar coordinates of the target), calculated in real-time using GPS and compass data.
- 2 fully-connected *hidden layers* of 32 neurons each, with *ReLU* activation functions.
- An *output layer* of 2 neurons, the first neuron indicates the linear velocity (i.e., the speed of the robot), and the second one provides the angular velocity (i.e., a single value indicating the rotation). These two actions can be executed simultaneously, providing the agent with richer movement options.

Training. The training of agents on this benchmark was also based on PPO (Schulman et al., 2017). However, unlike the previous case, here we employed a *continuous* reward function, given the increased complexity of this task:

$$R_t = \begin{cases} 1 & \text{the goal is reached} \\ -1 & \text{the agent collides} \\ (dist_{t-1} - dist_t) \cdot \eta - \beta & \text{otherwise} \end{cases}$$

where $dist_k$ is the distance from the target at time-step k; η is a normalization factor; and β is a penalty, intended to encourage the robot to reach the target quickly (in our experiments, we empirically set $\eta = 3$ and $\beta = 0.001$). Following is a list of hyperparameters employed during training:

- *training episodes*: 500
- *number of hidden layers*: 2
- *size of hidden layers*: 32
- *parallel environments*: 1
- *gamma (γ)*: 0.99
- *learning rate*: 0.0003
- *memory limit*: None
- *update frequency*: 1024 steps
- *trajectory reduction strategy*: sum
- *epochs*: 10
- *batch number*: 32
- *critic network size*: 2x256
- *PPO clip*: 0.2
- *GAE lambda*: 0.95
- *target kl-divergence*: 0.02
- *max gradient normal*: 0.5

Safety Requirements. The safety requirements for `Mapless Navigation` aim at guaranteeing the same objectives as the ones described for the `Particle World` environment. However, there is a crucial difference in this context: the consequences of an action may not always be predictable because the agent's increased degree of freedom results in a set of possible collision situations that cannot be detected by observation alone. For example, there may be an obstacle between two lidar scans that the agent cannot detect. Therefore, we cannot ensure the safety of the agent in *any* possible configuration, even if it meets all requirements. Our objective is thus to guarantee the agent's safety against the specified set of requirements, which may not encompass all potential collisions.

B Reactive Synthesis

In continuation to the temporal operators described in Sec. 2, additional temporal operators include $\mathcal{R}$ (*release*), $\Diamond$ (*finally*), and $\mathcal{W}$ (*weak until*) which can also be derived from the recursive syntax, e.g., $\varphi_0 \mathcal{R} \varphi_1 \equiv \neg(\neg\varphi_0 \,\mathcal{U}\, \neg\varphi_0)$. In addition, we note that equivalences are also well defined, i.e., distribuitivity properties (e.g., $\Box(\varphi_0 \wedge \varphi_1) \equiv (\Box\varphi_0) \wedge (\Box\varphi_1)$), negation properties (e.g., $\neg\Diamond \equiv \Box\neg\varphi$) and other temporal-specific properties (e.g., $\Box\varphi \equiv \Box\Box\varphi$).

Let ω denote infinite words (Wolfgang, 1990), then the semantics of LTL formulas associates traces $\sigma \in \Sigma^\omega$ with LTL fomulae (where $\sigma \models \top$ always holds, and $\vee$ and $\neg$ are standard):

$$
\begin{array}{ll}
\sigma \models a & \text{iff } \; a \in \sigma(0) \\
\sigma \models \bigcirc\varphi & \text{iff } \; \sigma^1 \models \varphi \\
\sigma \models \varphi_1 \,\mathcal{U}\, \varphi_2 & \text{iff } \; \text{for some } i \geq 0 \;\; \sigma^i \models \varphi_2, \text{ and for all } 0 \leq j < i, \sigma^j \models \varphi_1
\end{array}
$$

A safety formula φ is such that for every failing trace $\sigma \not\models \varphi$ there is a finite prefix u of σ, such that all σ' extending u also falsify φ, i.e., $\sigma' \not\models \varphi$. In this paper, we only synthesize models for safety formulae, which are indeed the most interesting ones for our problem and the fully monitorable ones.

Reactive LTL synthesis (Piterman et al., 2006; Thomas, 2008) is the task of producing a system that satisfies a given LTL specification φ, where atomic propositions in φ are split into variables controlled by the environment ("input variables") and by the system ("output variables"). Synthesis

corresponds to a game where, in each turn, the environment player produces values of the input propositions, and the system player responds with values of the output propositions. A play is an infinite sequence of turns, i.e., an infinite interaction of the system with the environment. A strategy for the system is said to be *winning* for the system if all the possible plays played according to the strategy satisfy the LTL formula φ.

C Symbolic Representation: Additional Details

In *fully observable* domains, it is possible to encode the environment with symbolic representations. This includes a representation of the arena as a formula ψ in propositional logic, which implies that its states can be precisely characterized as models of ψ. In other words, models $M = \{..., m_k, ...\} \neq \emptyset$ of ψ is a precise encoding of the arena, which means that each model can be obtained by performing classic Boolean satisfiability (SAT) queries over ψ, i.e., there exists a SAT encoding of the set of states.

As a toy example, let us consider an arena with four states that correspond to coordinates north-south and west-east: $\{\text{NE}, \text{NW}, \text{SE}, \text{SW}\}$. A formula $\phi = \text{NE} \vee \text{NW} \vee \text{SE} \vee \text{SW}$ encodes the whole arena, and a model of the formula is a concrete state. Moreover, we can encode groups of states of the arena using conjunctions. In the example above, the *north* group is encoded precisely by $\phi = \psi \wedge \neg(\text{SE} \vee \text{SW})$. These observations are very relevant for verification-guided shielding due to our ability to use such symbolic representations in order to precisely encode and represent disjunct safe and unsafe regions. .

Note that, since we are in a continuous domain, the amount of states is infinite, so this representation is not encoded with propositional logic, but rather with first-order logic modulo appropriate theories (in our case, linear real arithmetic (Monniaux, 2008)). Thus, we can obtain models from satisfiability modulo theory (SMT) queries, i.e., there is an SMT encoding of the set of states. We can modify the example above to show the difference. Consider the state is characterized by two input values of the environment, x_1 and x_2: *south* is represented by $x_1 : [0, 1)$ (respectively, $x_1 : [1, 2]$ represents *north*) and *west* is represented by $x_2 : [0, 1)$ (respectively, $x_2 : [1, 2]$ represents *east*). Then, the formula $\exists x_1, x_2.(0 \leq x_1 \leq 2) \wedge (0 \leq x_2 \leq 2)$ encodes all the infinite states in a succinct manner, i.e., states are models of this formula. Again, we can encode groups easily, e.g., it is possible to represent *south-east* with models of $\exists x_1, x_2.(0 \leq x_1 < 1) \wedge (1 \leq x_2 \leq 2)$, etc.

In summary, with symbolic encodings, we can compactly represent states (or regions) and sets of states and also simplify them. Note that these encodings are especially succinct if states have overlapping regions, i.e., share models in the SMT formula, since this allows the formula to be further simplified. In our empirical evaluation, we used Z3's (De Moura & Bjørner, 2008) `simplify(phi)` primitive for this step.

D Verification Example and Property Encodings

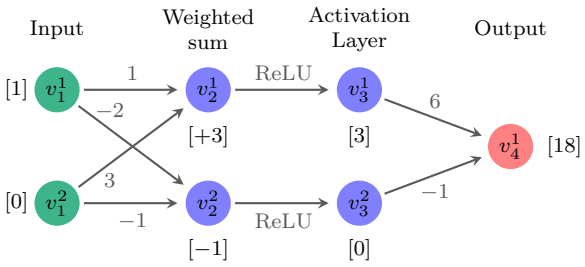

Figure 4: A toy DNN.

Suppose we wish to verify that the toy DNN depicted in Fig. 4 outputs, for any given input, a value strictly larger than 30, i.e., for any input $x = \langle v_1^1, v_1^2 \rangle$, the property $N(x) = v_4^1 > 30$ always holds. It is straightforward to encode this property as a verification query by using a precondition that does not restrict the inputs, i.e., $P = (true)$, and also, by setting $Q = (v_4^1 \leq 30)$ as a postcondition. Hence, for the verification query $P(x_0) \wedge Q(N(x_0))$, a sound verification engine will return `SAT`, along with a feasible counterexample, e.g., $x = \langle 1, 0 \rangle$, which produces $v_4^1 = 24 \leq 30$. Hence, proving that this property does not hold.

In this work, we used *Marabou* (Katz et al., 2019a; Wu et al., 2024) as our verification engine. Marabou is sound and complete and has recently been used in various applications (Elboher et al., 2020; Amir et al., 2021a;b; 2022; Refaeli & Katz, 2022; Elboher et al., 2022; Amir et al., 2023b;c).

Note. We note that similarly to previous work (Amir et al., 2023a), we typically considered violations of the required property, if the "wrong" action won by a given margin.

D.1 Verification Queries for `Particle World`

The general idea behind these properties is to ensure that if an obstacle is detected in one of the 4 possible directions, the agent will not take a step in that direction, and have the step size greater than the measured distance. The operator `argmax` encodes the fact that the agent can only move in one direction at a time, i.e., the one with the highest value; the constant 0.055 indicates the linear speed of the agent, which should be multiplied by the DNN's output action to obtain the actual step size. For example, if the DNN outputs $Y = [0.3, 0.2, 0.8, 0.0]$, the agent will move `Up` (i.e., the action associated to the third node) by $0.8 \cdot 0.055 = 0.044$ units. Below we report the complete formalization of the properties, where X is the input, Y is the output, $\mathcal{D}_x$ is the domain of the input space, and $\mathcal{N}$ is the neural network function; all inputs are normalized to the interval $[0, 1]$. Finally, we note that if the expression returns `true` (`SAT`), it means that there is an assignment that violates the properties, and the network is deemed unsafe.

- `Particle World 1 (G1)`: avoid collision with an obstacle on the **right** of the agent.
  - $(\texttt{argmax}(Y) == 0)$ and $(Y[0] \cdot 0.055 > X[0])$ and $(Y = \mathcal{N}(X))$ $\quad \forall_X \in \mathcal{D}_x$
- `Particle World 2 (G2)`: avoid collision with an obstacle on the **left** of the agent.
  - $(\texttt{argmax}(Y) == 1)$ and $(Y[1] \cdot 0.055 > X[1])$ and $(Y = \mathcal{N}(X))$ $\quad \forall_X \in \mathcal{D}_x$
- `Particle World 3 (G3)`: avoid collision with an obstacle **above** the agent.
  - $(\texttt{argmax}(Y) == 2)$ and $(Y[2] \cdot 0.055 > X[2])$ and $(Y = \mathcal{N}(X))$ $\quad \forall_X \in \mathcal{D}_x$
- `Particle World 4 (G5)`: avoid collision with an obstacle **below** the agent.
  - $(\texttt{argmax}(Y) == 3)$ and $(Y[3] \cdot 0.055 > X[3])$ and $(Y = \mathcal{N}(X))$ $\quad \forall_X \in \mathcal{D}_x$

D.2 Verification Queries for `Mapless Navigation`

The properties for the `Mapless Navigation` environment follow the same structure as explained for the previous benchmark. A crucial difference to note, which we already discussed in Appendix A, is that given the complex nature of the problem and the agent's high degree of freedom, we cannot guarantee the *absolute* safety of the agent. Hence, in this scenario, we settle instead on guaranteeing adherence to the following set of constraints.

- `Mapless Navigation 1 (M1)`: avoid collision with an obstacle **in front** of the robot.
  - $(X[3] - 0.17 < Y[0] \cdot 0.015)$ and $(Y = \mathcal{N}(X))$ $\quad \forall_X \in \mathcal{D}_x$
- `Mapless Navigation 2 (M2)`: avoid collision with an obstacle on the **left** of the robot.

- $(X[1] - 0.17 < Y[0] \cdot 0.015)$ and $(Y[1] < -0.2)$ and $(Y = \mathcal{N}(X))$ $\quad \forall_X \in \mathcal{D}_x$

- `Mapless Navigation 3 (M3)`: avoid collision with an obstacle *slightly* on the **left** of the robot.
  - $(X[2] - 0.17 < Y[0] \cdot 0.015)$ and $(Y[1] < -0.15)$ and $(Y = \mathcal{N}(X))$ $\quad \forall_X \in \mathcal{D}_x$
- `Mapless Navigation 4 (M4)`: avoid collision with an obstacle on the **right** of the robot.
  - $(X[5] - 0.17 < Y[0] \cdot 0.015)$ and $(Y[1] > 0.2)$ and $(Y = \mathcal{N}(X))$ $\quad \forall_X \in \mathcal{D}_x$
- `Mapless Navigation 5 (M5)`: avoid collision with an obstacle *slightly* on the **right** of the robot.
  - $(X[4] - 0.17 < Y[0] \cdot 0.015)$ and $(Y[1] > 0.15)$ and $(Y = \mathcal{N}(X))$ $\quad \forall_X \in \mathcal{D}_x$

E Formal Verification of Safe Regions

Algorithm 1 reports the pseudocode for step (2) of our approach, as described in Sec. 4. This subprocedure takes as input the approximated set of `SAT` regions $\tilde{S}$ (i.e., validated unsafe regions), and the approximated set of `UNSAT` regions $\tilde{U}$ (i.e., potentially safe regions). The algorithm iterates over $\tilde{U}$, while verifying the regions with a formal verification tool (e.g., `Marabou` (Katz et al., 2019a)), to formally ensure that these regions are actually safe. If the result is `SAT`, we relabel the region in question as unsafe. After the process, all regions in the `UNSAT` set are formally safe. This ensures that decisions made by the policy in these regions are reliable without the need for shielding.

Algorithm 1 Formal verification of safe regions.

Require: $\tilde{\mathcal{S}}, \tilde{\mathcal{U}}$
Ensure: $\mathcal{S}, \mathcal{U}$
1: $y \leftarrow 1, \mathcal{S} \leftarrow \emptyset, \mathcal{U} \leftarrow \emptyset$
2: **for** region **in** $\tilde{\mathcal{U}}$ **do**
3: **if** `formal-verificaiton(region)` **is** `SAT` **then** ▷ call a verification tool backend
4: `remove(region,` $\tilde{\mathcal{U}}$`)`
5: `add(region,` $\mathcal{S}$`)`
6: **end if**
7: $\mathcal{U} \leftarrow \tilde{\mathcal{U}}, \mathcal{S} \leftarrow \tilde{\mathcal{S}} \cup \mathcal{S}$
8: **end for**