LEARNING OBLIQUE ABSTRACTIONS FOR THE SAFETY VERIFICATION OF OBLIQUE DT POLICIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep Reinforcement Learning (DRL) has shown major success. In safety-critical systems, though, a question mark remains on the safety of policies learned by DRL. In this paper, we consider the verification of classical reach-avoid properties for DRL problems. Our approach is the following: i) Consider Oblique Decision Trees (ODTs) as policies, which are more efficient than axis-aligned DT policies, ii) Learn unions of polytopes as abstractions of the sets reached after n steps for all n till goal is reached (reach-avoid), or as an invariant (pure safety), iii) Verify these candidate abstractions or invariants formally. The main tools we leverage are neurosymbolic architectures encoding ODTs as differentiable neural networks. They have already been shown to generate small ODT policies as efficient as pure Deep RL for Cyber-Physical Systems. The novelty here is to equip them with dedicated loss functions, to learn abstractions and invariants in the form of unions of polytopes. We demonstrate our techniques on *discrete actions, discrete time* environments MountainCar and Quadrotor (reach-avoid) as well as CartPole (safety). The number of polytopes to verify the policy is reduced by 1-2 orders of magnitude compared with using axis-aligned polytopes.

1 INTRODUCTION

Deep reinforcement learning (DRL) (Mnih et al., 2015; Lillicrap et al., 2015) has demonstrated promising performance across a broad range of domains, particularly in the control of cyber-physical systems (CPS), where sensory inputs are mapped to control actions. However, deploying DRL-based learned controllers in CPS, many of which are safety-critical, poses significant risks. A policy that performs well on average may still drive the system into unsafe states. Recently, there has been growing interest in providing formal guarantees on the closed-loop behavior of learned controllers. Two classes of properties are of particular interest. A *reach-avoid* property requires that every trajectory starting from a given set of initial states eventually reaches a target set while avoiding a set of unsafe states along the way. A *safety* property requires that the system never enters an unsafe state, over an unbounded time horizon. Several works provide soft, statistical guarantees for such properties, for instance by bounding the probability of failure through sampling-based evaluation (Legay et al., 2010; Bacci & Parker, 2020; Katz et al., 2023). Such guarantees, however, do not rule out rare but potentially catastrophic trajectories. The policy may still produce undesirable trajectories that occur with low probability and are therefore unlikely to be observed during testing. This has motivated growing interest in providing hard guarantees that hold for *all* trajectories of the closed-loop system (Ivanov et al., 2019; Tran et al., 2020; Eliyahu et al., 2021; Amir et al., 2021; Schilling et al., 2023).

Hard guarantees are typically obtained through reachability analysis (Althoff et al., 2021). Since the set of states reachable from an initial set cannot, in general, be computed exactly, these methods compute a sound over-approximation of it, known as an *abstraction*. Starting from an abstract representation of the initial set, the abstraction is propagated step by step through the policy and the system dynamics, producing a sequence of sets that together contain every reachable state. This sequence is commonly called a *flowpipe* (Frehse et al., 2011). Abstract interpretation (Cousot, 2021) formalizes this process by fixing an abstract domain, such as intervals (hyperrectangles), zonotopes (Girard, 2005), or polytopes (Singh et al., 2017b), together with sound transfer functions over that domain. A reach-avoid property is established if every set in the flowpipe avoids the unsafe region and the flowpipe eventually lies within the target set. Overly coarse abstractions, however, accumulate

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

over-approximation error over successive steps, known as the wrapping effect (Neumaier, 1993), and may lead to spurious violations. Pure safety properties are more demanding, since they must hold over an unbounded horizon and cannot be established by a finite flowpipe alone (unless the abstract flowpipe is contracting, not the case in general). For such properties, (forward) invariants offer an alternative. That is, a set of states containing the initial states and away from the unsafe state that cannot be evaded by the closed-loop system. Such invariants are however notoriously hard to obtain.

Within this line of research, a substantial body of work focuses on neural network (NN) policies (Dutta et al., 2018; Ivanov et al., 2019; Tran et al., 2020; Huang et al., 2022). In this setting, an NN is trained within a DRL framework to represent the policy and is then subjected to formal verification. While verifying simple properties of NNs is possible (Katz et al., 2017; Eliyahu et al., 2021; Amir et al., 2021), the difficulty is compounded in the closed-loop setting, where the network must be reasoned about jointly with the system dynamics over multiple time steps. To overcome this limitation, several studies (Bastani et al., 2018; Schilling et al., 2023) represent the policy using decision trees (DTs), which are more amenable to verification because they can be described with few pieces (polytopes). They use axis-aligned DTs (AADTs), with decision nodes splitting on single features, leading to hyperrectangles abstractions. On the other hand, oblique DTs (ODTs) (Karthikeyan et al., 2022; Panda et al., 2026) split on linear combinations of state variables, leading to general polytopes that are significantly more expressive/efficient (Silva et al., 2020; Kohler et al., 2024; Lopez-Miguel et al., 2026; Panda et al., 2026). Most existing abstraction methods rely on abstract interpretation over fixed domains (Schilling et al., 2023; Ranzato & Zanella, 2020), which cannot efficiently represent faithfully general polytopes without a number of hyperrectangles exponential with the dimension.

Polytope abstractions (Cousot, 2021; Sankaranarayanan et al., 2005), whose facets can align with both the decision boundaries of the policy and the geometry of the reachable set raise two challenges. First, it is unclear how to choose these polytopes, as no fixed set of templates is adequate across environments (Bogomolov et al., 2017; Entesari & Fazlyab, 2023). Second, propagating the possibly nonlinear, system dynamics to obtain the polytope abstraction at the next time step while preserving soundness and precision is nontrivial (Cousot, 2021; Singh et al., 2017b; 2019).

In this paper, we address both challenges by devising a novel neurosymbolic framework to prove the safety of systems controlled by oblique DT policies, for both reach-avoid and pure safety properties. The key idea is to *learn* highly accurate general polytope abstraction *candidates* for the flowpipe or invariants of the closed-loop system, and then verify the candidates. Union of polytopes are equivalent to ODTs, so we adapt DTSemNet (Panda et al., 2026), an efficient architecture that encodes an ODT as a NN amenable to gradient-based training.

Contributions. Our contributions can be summarized as follows:

- For reach-avoid properties, we propose a novel **Ne**uro**Sy**mbolic **Verif**ication (NeSyVerif) technique to learn union of polytopes as abstraction candidates, verify them symbolically, with a feedback loop if the verification returns a counterexample. Technically, DTSemNet learns an ODT with a fixed architecture. We extend the learning framework to support adaptive growth and pruning, which is necessary for learning accurate abstractions. We also introduce dedicated loss functions that match the conditions required for the abstractions.
- For safety properties, we show that unions of polytopes are complete at representing invariants for robustly-safe ODT policies. This is similar to the converse Lyapunov theorem, though our result works with controlled systems with discontinuities, whereas the proof of the converse Lyapunov theorem relies on the pure continuity of the system under control. To find these unions of polytopes as invariant candidates, we devise dedicated loss functions.
- We demonstrate the effectiveness of our approach by verifying safety and reach-avoid properties for several oblique DT policies trained with DRL on a range of CPS environments. Compared to using axis-aligned abstractions of AADT policies, the number of polytopes of the abstractions needed to verify the policy is lowered by 1 to 2 orders of magnitude.

2 Related Work

Verification of Learned Controllers. Closed-loop verification of NN policies alternates between reasoning about the network and propagating sets through the dynamics, using intervals, polytopes, star sets, or Taylor models (Dutta et al., 2018; Ivanov et al., 2019; Tran et al., 2020; Huang et al.,

2022). Its scalability is limited by the hardness of NN verification (Katz et al., 2017; Eliyahu et al., 2021; Amir et al., 2021). DT policies offer a more tractable alternative (Bastani et al., 2018). Schilling et al. (2023) verify DT-controlled systems by propagating set-based approximations through the decision nodes. Like abstract interpretation approaches for tree ensembles (Ranzato & Zanella, 2020), however, they rely on axis-aligned abstractions. Oblique DTs are often needed to match the performance of deep RL policies (Lopez-Miguel et al., 2026; Panda et al., 2026), and they can exactly represent piecewise-affine control laws such as linear MPC (Ren et al., 2025). Yet, to our knowledge, no prior work verifies closed-loop systems under oblique DT policies using oblique abstractions aligned with their decision boundaries.

Neural Control with Certificates. Certificate functions, such as Lyapunov (Khalil & Grizzle, 2002) or barrier functions (Prajna & Jadbabaie, 2004), provide another classical approach to verifying controllers. Early works consider polynomial control systems and compute polynomial certificate functions via constrained programming (Henrion & Garulli, 2005; Parrilo, 2000; Jarvis-Wloszek et al., 2003). More recent works consider *neural* certificate functions for verifying neural controllers. The neural certificate is learned and formally verified through a counterexample-guided inductive synthesis (CEGIS) loop, with verification achieved either through SMT or MILP solving (Chang et al., 2019; Zhao et al., 2020; Abate et al., 2021b;a), or Lipschitz analysis and bound propagation (Lechner et al., 2022; Zikelic et al., 2023a;b; Neustroev et al., 2025; Badings et al., 2025). However, these works are restricted to control tasks with *continuous* action spaces. In contrast, in this work we use DT policies and oblique abstractions, and consider control tasks with *discrete* action spaces.

Polytope Abstractions. General polyhedra are the most expressive numerical domain but using it for reachability is challenging (Singh et al., 2017b;a). This motivates template polyhedra (Sankaranarayanan et al., 2005), in which facet directions are fixed and abstract operations reduce to linear programming. Precision then depends on the choice of template. Prior work refines template directions using counterexamples (Bogomolov et al., 2017), specifies them manually to prove infinite-horizon safety of NN policies (Bacci et al., 2021), or derives them analytically from the structure of the network (Entesari & Fazlyab, 2023). We instead learn polytopic flowpipes directly from the sampled data of closed-loop system and certify them symbolically.

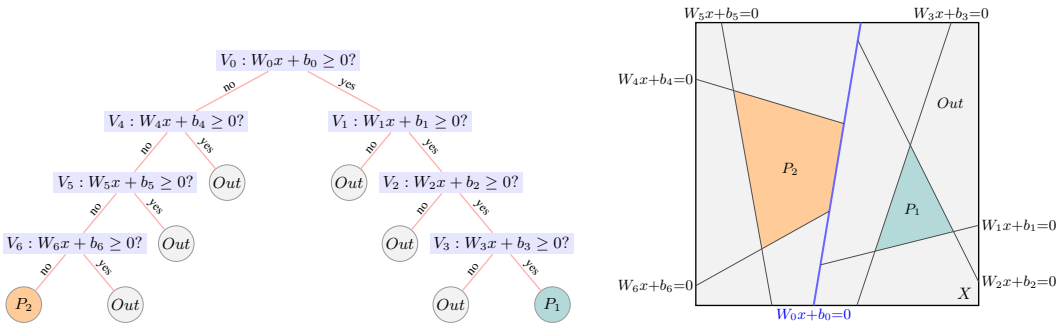
3 PRELIMINARIES

For a Euclidean space $\mathbb{R}^n$, we use $||.||$ to denote the L_∞-norm. For $\varepsilon > 0$ and a set $Z \subseteq \mathbb{R}^n$, we define $\mathcal{B}_\varepsilon(Z) = \{x' \in \mathbb{R}^n \mid \exists x \in Z. ||x' - x|| < \varepsilon\}$ to be the open ball of radius ε around Z, and write $\mathcal{B}_\varepsilon(x)$ for $\mathcal{B}_\varepsilon(\{x\})$. A set-valued function $f : Z \to 2^{\mathbb{R}^m}$ with $Z \subseteq \mathbb{R}^n$ is said to be Lipschitz continuous with a Lipschitz constant K, if for every $x, x' \in Z$ and every $y' \in f(x')$, there exists $y \in f(x)$ with $||y' - y|| \leq K \cdot ||x' - x||$; for single-valued f, this is the usual condition $||f(x') - f(x)|| \leq K \cdot ||x' - x||$. Notation is summarized in Appendix A.

System. A *discrete-action control system* (or, simply, a *system*) S is a 4-tuple $(X, A, succ, I)$, where (1) $X \subseteq \mathbb{R}^n$ is the set of *states*, (2) A is the discrete set of *control actions*, (3) $succ : X \times A \to 2^X \setminus \emptyset$ is the *successor function* which is assumed to be Lipschitz continuous in the first argument and provides the set $succ(x, a)$ of possible successor states when selecting action a in state x, and (4) $I \subseteq X$ the set of *initial states*. We assume that the successor function is given as Runge-Kutta RK4 equations. Notice that, although the successor function is Lipschitz-continuous *for a given action*, a controller will usually use different actions at different states. At the frontier between two actions, there will be discontinuities in general.

Policy. A *policy* σ for system $S = (X, A, succ, I)$ is a function $\sigma : X \to A$ which to each state prescribes an action. Given a policy σ, we define $succ_\sigma : X \to 2^X$ by $succ_\sigma(x) = succ(x, \sigma(x))$. A *run* under σ is a sequence $(x_i)_{i \in \mathbb{N}}$ of states such that $x_0 \in I$ and $x_{i+1} \in succ_\sigma(x_i)$ for all $i \geq 0$. We define $Reach^S(\sigma) = \{x \mid \exists k \in \mathbb{N}. (x_i)_{i \in \mathbb{N}}$ a run under σ and $x = x_k\}$ to be the set of all reachable states, and $Run^S(\sigma)$ to be the set of all runs in S under policy σ.

Perturbed System. For a system $S = (X, A, succ, I)$, we define its δ-perturbation $S_\delta = (X, A, succ_\delta, I)$ via $succ_\delta(x, a) = \cup_{x' \in \mathcal{B}_\delta(x)}(succ(x', a))$. Note that each policy σ in S also induces a policy for S_δ, and we define $Reach^S_\delta(\sigma) = Reach^{S_\delta}(\sigma)$. When S and σ are clear from the context, we write $\mathcal{R}_\delta = Reach^S_\delta(\sigma)$ for the set of states reachable in the δ-perturbed system.



(a) ODT encoding two polytopes P_1, P_2.

(b) Representation of P_1, P_2 in X.

Figure 1: (a) ODT $\mathcal{T}$ over 3 classes (P_1, P_2, Out) (b) Associated partition of X into polytopes.

Reach-Avoid and Safety Specifications. In this work, we consider the problem of verifying that a policy in a system satisfies one of the two classical properties: reach-avoid and safety specifications. A *reach-avoid specification* is defined by a set $R \subseteq X$ of target states to be reached, and a set $U \subseteq X$ of unsafe states to be avoided. A policy σ is said to *satisfy* a reach-avoid specification if, for every run $(x_i)_{i\in\mathbb{N}} \in Run^S(\sigma)$, there exists a time step $k \in \mathbb{N}$ such that $x_k \in R$ and $x_i \notin U$ for all $i \leq k$.

A *safety specification* is defined by a set of unsafe states $U \subseteq X$. A policy σ is said to *satisfy* a safety specification if no run under σ reaches an unsafe state, i.e. $Reach^S(\sigma) \cap U = \emptyset$. Given $\delta > 0$, a policy σ is said to be *δ-robustly safe*, if it satisfies the safety specification in the perturbed system S_δ. Finally, it is said to be *robustly safe*, if it is δ-robustly safe for some $\delta > 0$.

Oblique Decision Trees. An *Oblique Decision Tree (ODT)* $\mathcal{T} = (V, L, \text{children}, (W_i, b_i)_{0\leq i\leq |V|-1}, \text{class})$ over state space X and a finite set of classes $\mathcal{Y}$ (see Fig. 1a for an example) is a tuple, where:

- V is the set of *internal nodes*, L is the set of *leaf nodes*, and $V \sqcup L$ is the set of nodes with root $\in V \sqcup L$ the root of the tree.
- children : $V \to ((V \sqcup L) \setminus \{\text{root}\})^2$ maps each internal node to a pair of children nodes. For each internal node $i \in V$, children$(i) = (j, k)$ with $j \neq k$. For each $j \in (V \sqcup L) \setminus \{\text{root}\}$, there exists exactly one $i \in V$ so that either children$(i) = (j, k)$ or children$(i) = (k, j)$.
- Each internal node $i \in V$ is associated an affine inequality $W_ix + b_i \geq 0$, $W_i \in \mathbb{R}^n, b_i \in \mathbb{R}$.
- class : $L \to \mathcal{Y}$ associates with each leaf node a class in $\mathcal{Y}$ (e.g. $\mathcal{Y} = \{0, 1\}$ or action set A).

An *Axis-Aligned DT (AADT)* is an ODT where, for each internal node $i \in V$, the vector W_i has a unique non-zero coefficient. For an input state $x \in X$, the ODT $\mathcal{T}$ computes the *class output* $\mathcal{T}_x$ in $\mathcal{Y}$ as follows. The computation starts from the root of the tree. At every step, if the computation is in internal node $i \in V$ with children$(i) = (j, k)$, the computation evaluates the inequality $W_ix + b_i \geq 0$ associated to node i. If the inequality is satisfied, the computation moves to node j. Otherwise, it moves to node k. Once a leaf node $l_x \in L$ is reached, the class output is defined by $\mathcal{T}_x = \text{class}(l_x)$.

Note that each leaf l in $\mathcal{T}$ is associated with a *polytope* $P_l = \{x \in X \mid \forall i \in \text{path}(l).\, W_ix + b_i \bowtie_i 0\}$, where path$(l)$ is the path from the root to the leaf l in $\mathcal{T}$, and each $\bowtie_i \in \{\geq, <\}$ depending on the child node along the path. Thus, an ODT partitions the state space X into $|L|$ polytopes, see Fig. 1b for an example. In the case of AADTs, each inequality has a single non-zero linear coefficient, so AADT leaf nodes partitin the state space X into axis-aligned *hyperrectangles*.

Problem Statement. We now define the problem that we consider in this work. Consider a system $S = (X, A, succ, I)$, an ODT policy $\sigma : X \to A$, and a reach-avoid or safety specification. A policy σ is an *ODT policy*, if there exists an ODT $\mathcal{T}$ over state space X and the class set of actions A such that $\sigma(x) = \mathcal{T}_x$ for all $x \in X$. Our goal is to *verify* that the ODT policy σ satisfies the specification.

4 Neurosymbolic Verification for Reach-Avoid Specifications

We now present our **Ne**uro**Sy**mbolic **Verif**ication (**NeSyVerif**) framework for verifying ODT policies with respect to reach-avoid specifications. Let $S = (X, A, succ, I)$ be a system, $\sigma : X \to A$ be an ODT policy, and consider a reach-avoid specification defined by the set of target states $R \subseteq X$ and

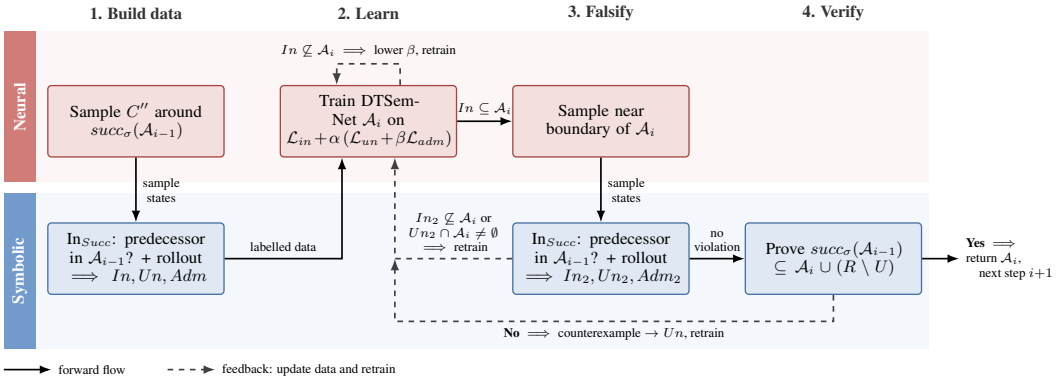


Figure 2: **NeSyVerif** framework for learning the abstraction $\mathcal{A}_i$ of $succ_\sigma(\mathcal{A}_{i-1})$ (Section 4). Neural components (top lane) get boundary samples and train the DTSemNet classifier; symbolic components (bottom lane) correctly labels the sample and formally verify $\mathcal{A}_i$. $\mathcal{L}_{in}$ forces every states in In inside $\mathcal{A}_i$, $\mathcal{L}_{un}$ keeps unsafe states out of $\mathcal{A}_i$, and $\mathcal{L}_{adm}$ keeps as few admissible configurations as possible in $\mathcal{A}_i$ (as tight as possible abstraction). If some $c \in In$ is misclassified, training restarts with a smaller β. Counterexamples from the boundary check or from verification are added to the training data and $\mathcal{A}_i$ is retrained. For an ODT-certificate, the same loop is used with $\mathcal{A}_{i-1} = \mathcal{A}_i = B$ (Section 5).

the set of unsafe states $U \subseteq X$. **NeSyVerif** inductively computes a sequence of abstractions $(\mathcal{A}_i)_{i\geq 0}$ such that $\mathcal{A}_0 = I$ is the set of initial states, and $succ_\sigma(\mathcal{A}_{i-1}) \subseteq \mathcal{A}_i \cup (R \setminus U)$ for all $i \geq 1$. This ensures that $\mathcal{A}_i$ contains reachable states at time steps i along all runs that have not yet satisfied the reach-avoid specification. The computation stops when $\mathcal{A}_i \subseteq R \setminus U$, which implies that the reach-avoid specification is satisfied for all runs. The abstractions $\mathcal{A}_i$ are trained from data and symbolically verified, and we describe the procedure below.

Training Data. The abstractions $\mathcal{A}_i$ are trained inductively. We initialize $\mathcal{A}_0 = I$. Suppose that $\mathcal{A}_{i-1}$ was already learned and that we want to train $\mathcal{A}_i$. First, we collect training datea for $\mathcal{A}_i$ as follows. We sample states C inside $\mathcal{A}_{i-1}$, and we sample $C' = succ_\sigma(C)$. We then set up a bounding box Box strictly containing all states in C' (with some margins), and sample states C'' inside Box. For each state $c'' \in C''$, we check whether there exists a state c with $c'' = succ_\sigma(c)$, using a symbolic procedure In_{Succ} described below. If the answer is positive, we tag c'' as *inside*. If inside states are too close to the boundary of Box, we enlarge Box.

Next, for each $c'' \in C''$, we iteratively sample its successor states for a number of time steps which is an algorithm hyperparameter to catch if a run from c'' violated the reach-avoid specification, i.e. reaches U without reaching R. If the sampled run from c'' is found to violate the reach-avoid specification, we tag c'' as *unsafe*. Otherwise, we tag it as *admissible* if it was not tagged as *inside*.

If any state $c'' \in C''$ was tagged both *inside* and *unsafe*, we backtrack and repeat the procedure. Otherwise, we end up with three disjoint sets of training data states In, Un and Adm that were tagged *inside*, *unsafe* and *admissible*, respectively.

In_{Succ}: Given state c'', the *Symbolic predecessor computation* consists of three components:

- **Finding constants:** We first consider a ρ-cube around c''. Over this cube, we derive a certified range for the inverse Jacobian matrix of the transition map. This is possible due to the RK4 representation. We then verify that the determinant of the inverse Jacobian matrix is bounded below by a positive constant $\lambda > 0$ throughout the certified range, thereby establishing the local properties required for the subsequent verification.
- **Heuristic candidate generation.** Newton's method is employed to solve the RK4 equations backward and iteratively refine a candidate predecessor c. The iteration terminates only when the distance d between c'' and $succ_\sigma(c)$ is sufficiently smaller than $\min\{\lambda\rho, \frac{1}{\lambda}\rho\}$. Since Newton's method is used solely for candidate generation, its success or failure does not affect the soundness of the algorithm.
- **Verification:** Once a candidate predecessor is found, we verify that it belongs to the current abstraction and that its distance to the abstraction boundary exceeds the threshold determined by λ and d. If both conditions are satisfied, c'' is soundly tagged as *inside*.

Training Loss. We train the abstraction $\mathcal{A}_i$ as an ODT with two output classes $\mathcal{Y} = \{0, 1\}$. $\mathcal{A}_i(x) = 0$ means that the state is outside $\mathcal{A}_i$, whereas $\mathcal{A}_i(x) = 1$ means that the state is inside $\mathcal{A}_i$. We use the DTSemNet differentiable ODT encoding of (Panda et al., 2026)) for training. The loss function that the training algorithm minimizes is obtained by aggregating three loss terms $\mathcal{L} = \mathcal{L}_{in} + \alpha(\mathcal{L}_{un} + \beta\mathcal{L}_{adm})$, with $\alpha, \beta \leq 1$ being hyperparameters, and each loss term defined as follows:

- $\mathcal{L}_{in} = \sum_{c \in In} -\log p_1(c)$, encouraging states in In to have greater p_1 values.
- $\mathcal{L}_{un} = \sum_{c \in Un} -\log\big(1 - p_1(c)\big)$, encouraging states in Un to have smaller p_1 values.
- $\mathcal{L}_{adm} = \sum_{c \in Adm} -\log\big(1 - p_1(c)\big)$, encouraging states in Adm to have smaller p_1 values.

Here, $p_1(c)$ denotes the model's predicted probability (trained with softmax activation function) of a state being classified as inside $\mathcal{A}_i$. The state c is classified as inside $\mathcal{A}_i$ if $p_1(c) > 0.5$ (class 1), and classified as outside (class 0) otherwise.

Intuitively, the first loss term $\mathcal{L}_{in}$ encourages states in In to be inside $\mathcal{A}_i$, as otherwise the abstraction would not be inductive. The second loss term $\mathcal{L}_{un}$ encourages states in Un to be outside of $\mathcal{A}_i$, as runs from them violate the reach-avoid specification. Finally, the third loss term $\mathcal{L}_{adm}$ encourages states in Adm to be outside of $\mathcal{A}_i$. While this is not a strict requirement and would not violate inductiveness or reach-avoid satisfaction, we use this term to make the abstraction as tight as possible. That is why the term has an additional scaling hyperparameter $\beta \leq 1$.

Tuning hyperparameters α, β is not simple, as it depends on In, Un, Adm. To do so, we design an automatic tuning procedure, starting at $\alpha = 1$ and $\beta = 0.5$. We then lower their values every E epoch (hyperparameter: 10-50) until all In are well classified. The rationale is the following: we can prove that there exists α small enough so that for an abstraction $\mathcal{A}$ minimizing the loss, all states in In are inside $\mathcal{A}$. Proof is in Appendix B.

Proposition 1. *Let $\mathcal{L}_\alpha = \mathcal{L}_{in} + \alpha(\mathcal{L}_{un} + \beta\mathcal{L}_{adm})$, with $\beta < 1$. There exists $\bar{\alpha} > 0$ such that for all $\alpha < \bar{\alpha}$, considering an abstraction $\mathcal{A}_\alpha$ minimizing the loss $\mathcal{L}_\alpha$, we have $In \subseteq \mathcal{A}_\alpha$.*

Adaptive growth of the ODT Architecture: Gradient-based ODT learning (e.g. DTSemNet and DGT) typically learn a fixed ODT architecture. We thus design a heuristic to grow the ODT architectures to obtain a tighter abstraction $\mathcal{A}_i$. The initial ODT architecture we consider corresponds to a unique polytope P with a reduced number of equations (e.g. $8\times$ number of dimensions). We first refine P to exclude admissible samples till improvement is negligeabe. Then we partition P in 2 polytopes using architecture in Fig. 1a, and proceed inductively on each, till improvement is negligeabe. Arbitrary number of equations and polytopes can be obtained (Appendix B.2).

Symbolic Verification. We verify $succ_\sigma(\mathcal{A}_{i-1}) \subseteq \mathcal{A}_i \cup (R \setminus U)$ by proving the stronger containment $succ_\sigma(\mathcal{A}_{i-1}) \subseteq \mathcal{A}_i$ with LP/MILP:

- Variables encode the current state $x \in P_k \cap Q_j$, intermediate states (RK4 stages when used), nonlinear terms, and the successor x'. Here P_k is a polytope of $\mathcal{A}_{i-1}$, and Q_j is a policy leaf with fixed action a_j. Constraints encode one step under a_j.
- For $y = f(z)$, we add affine bounds $\ell_f(z) \leq y \leq u_f(z)$ on a certified interval. For convex f, tangents give lower and secants upper bounds (reversed for concave f); bilinear products use McCormick envelopes (McCormick, 1976). The constraints thus contain every exact transition.
- Write $\mathcal{A}_i = \bigcup_l P_l$, with $P_l = \{x' : W_{l,r}x' + b_{l,r} \geq 0 \text{ for all } r\}$. We search for x' satisfying $\forall l\ \exists r :\ W_{l,r}x' + b_{l,r} < 0$. For one P_l, this gives one LP per inequality; for a union, binary variables select one violated inequality per P_l, giving an MILP (Appendix C).
- A feasible relaxed query gives a potential counterexample $x \in \mathcal{A}_{i-1}$. We compute its exact successor $succ(x, a_j)$; only a successor outside $\mathcal{A}_i$ is used for retraining.
- If all relaxed queries are infeasible, containment holds. We also require $\mathcal{A}_i \cap U = \emptyset$.

Proposition 2 (Soundness of NeSyVerif)**.** *Suppose that **NeSyVerif** computes abstractions $\mathcal{A}_0, \ldots, \mathcal{A}_{N-1}$ with $\mathcal{A}_0 = I$ and $N \geq 1$. Assume that $\mathcal{A}_i \cap U = \emptyset$ for $0 \leq i < N$, that*

$$succ_\sigma(\mathcal{A}_{i-1}) \subseteq \mathcal{A}_i \cup (R \setminus U) \quad \textit{for } 1 \leq i < N, \textit{ and that } succ_\sigma(\mathcal{A}_{N-1}) \subseteq R \setminus U$$

Then the ODT policy σ satisfies the reach-avoid specification within N steps.

The proof is given in Appendix B.3.

5 NEUROSYMBOLIC TECHNIQUE FOR SAFETY SPECIFICATIONS

Definition 1. A safety certificate for policy σ of system $S = (X, A, succ, I)$ for unsafe region U is a set B such that:

1. $I \subseteq B$ (all initial states are in B),
2. $U \cap B = \emptyset$ (no unsafe state is in B), and
3. $succ_\sigma(x) \subseteq B$ for every $x \in B$ (B is forward invariant).

We consider B provided as an ODT over classes $\{0, 1\}$ (in/out), i.e. B is the finite union of polytopes classified as class 1; such a safety certificate is called an *ODT-certificate*.

One main question with ODTs is whether unions of polytopes are expressive enough to depict safety-certificate for σ. We answer positively for ODT-policies σ that are *robustly-safe* for some δ-perturbation (recall the definition in Section 3). This last hypothesis is similar to hypothesis for the converse Lyapunov theorem Khalil & Grizzle (2002), which holds for stability in purely continuous systems, We consider safety for systems with discontinuities, which break the proof for the converse Lyapunov theorem. The proof can be found in Appendix B.4.

Theorem 2. *Let $S = (X, A, succ, I)$ be a system, $U \subseteq X$ an unsafe set, and σ an ODT-policy that is* robustly-safe *for U. Then there is an ODT-certificate for σ of S for U.*

Learning the ODT-certificate. Let σ be an ODT-policy for an environment with safety specification U. Assume that σ is δ-robustly safe wrt unsafe set U. We want to formally prove the safety of σ. For that, we will learn an ODT-certificate using differentiable trees, e.g. Panda et al. (2026).

Considering the proof of Theorem 2, we will look for an ODT-certificate B such that:

1. The set $\mathcal{R}_\delta$ reachable from I under perturbation δ is inside B, that is $\mathcal{R}_\delta \subseteq B$.
2. there is no unsafe states in B, i.e. $U \cap B = \emptyset$,
3. if $c \in B$, then $succ_\sigma(c) \subseteq B$ (under no perturbation).

Training Data. We start by sampling states of $\mathcal{R}_\delta$; with a slight abuse of notation, we also write $\mathcal{R}_\delta$ for this finite sample. We then sample a set Adm of states around $\mathcal{R}_\delta$, which are initially labelled *admissible*, and we set $Un = \emptyset$.

We refine the ODT-certificate B inductively, by splitting polytopes in 2 or adding more equations. Once a candidate ODT-certificate B has been learnt, let $p_1(c)$ denote the model's predicted probability for sample c (in training with softmax activation function) of class 1 (inside). Each time we refine the candidate B, we sample a set C_2 of states inside and close to the boundaries, that is with $p_1(c) \in [0.5, 0.6]$, and, for each state $c \in C_2 \cap B$, a set $\mathcal{S}_c$ of successors in $succ_\sigma(c)$ (under no perturbation). A state c with $\mathcal{S}_c \not\subseteq B$ is *unsafe* and added to Un; the other states of C_2 are *admissible* and added to Adm. The sets Un and Adm play roles analogous to, but different from, those in Section 4: here, they are defined with respect to the forward invariance of the current candidate B rather than with respect to a sampled rollout.

Losses. For each iteration, we train an ODT with the following losses, with two classes {0,1}.

- $\mathcal{L}_{in} = \sum_{c \in \mathcal{R}_\delta} -\log p_1(c)$, encouraging states in $\mathcal{R}_\delta$ to have greater p_1 values.
- $\mathcal{L}_{un} = \sum_{c \in Un \cup U} -\log\big(1 - p_1(c)\big)$, encouraging states in $U \cup Un$ to have smaller p_1 values.
- $\mathcal{L}_{adm} = \sum_{c \in Adm} -\log\big(1 - p_1(c)\big)$, encouraging states in Adm to have smaller p_1 values.
- $\mathcal{L}^{next}_{adm} = \sum_{\substack{c \in Adm \\ p_1(c) \in [0.5, 0.6]}} \frac{1}{|\mathcal{S}_c|} \sum_{s \in \mathcal{S}_c} \max\big(0,\ \log p_1(c) - \log p_1(s)\big)$: if $p_1(c) \geq 0.5$, that is c is inside the current B, then there is a penalty for a successor $s \in \mathcal{S}_c$ having $p_1(s) < p_1(c)$, i.e. when s is closer to outside than c. That is, we encourage contraction within B for states c inside B close to the boundaries $p_1(c) \in [0.5, 0.6]$.

The first three losses mirror those used for abstraction learning in the reach-avoid case. The last one ensures that the ODT-certificate B learnt is compatible with the physics: indeed, states c in B with $s \in succ_\sigma(c)$ out of B will in the next step be in the unsafe set Un, and will need to be removed.

The join loss we consider is similar to what we used for reach-avoid:

$$\mathcal{L}_{in} + \alpha(\mathcal{L}_{un} + \mathcal{L}^{next}_{adm} + \beta\mathcal{L}_{adm}),$$

Table 1: Axis-Aligned type: AADT-policy verified using VeriXAI and axis-aligned abstractions. Oblique type: ODT-policy verified using NeSyVerif learning oblique abstractions and ODT-certificates. #Leaves reports the size of the policy, Reward for the efficiency of the policy, and #Polytopes the number of polytopes built by VeriXAI/NeSyVerif to verify them. **Bold**: better result.

Environment:	MountainCar		QuadRotor		CartPole	
Type:	Axis-Aligned	Oblique	Axis-Aligned	Oblique	Axis-Aligned	Oblique
#Leaves ↓	**2**	3	86	**8**	3	**2**
Reward ↑	-126.32 ± 1.47	$\mathbf{-109.95} \pm 0.43$	$\mathbf{-30}$	$\mathbf{-30}$	407.54 ± 36.75	$\mathbf{499.78} \pm 1.99$
#Polytopes ↓	1481	**129**	8241	**84**	136	**2**

that is with $\mathcal{L}_{adm}^{next}$ at the same level as $\mathcal{L}_{un}$, as sample in penalized by $\mathcal{L}_{adm}^{next}$ will be later penalized by $\mathcal{L}_{un}$. We keep the same induction on ODT-architecture as for reach-avoid specification.

Verification Once a candidate ODT-certificate has been learnt, we push it to the verifier, similarly as for Section 4. Counterexamples found serve for retraining. Details can be found in Appendix C.

6 Experiments

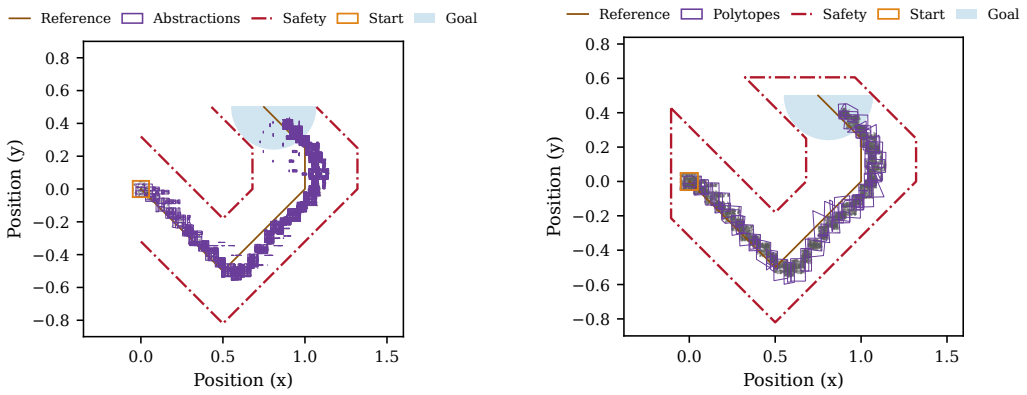
We evaluate whether ODT-policies, which achieve higher reward with fewer leaves, combined with oblique abstractions, reduce the number of polytopes needed to represent flowpipes and ODT-certificates compared with AADT-policies with axis-aligned abstractions.

Benchmarks and baselines. We consider three environments: MountainCar and QuadRotor (reach-avoid), and CartPole (safety). Appendix D describes the dynamics, specifications, and policies. We compare against VeriXAI[1] (Schilling et al., 2023), which verifies AADT-policies obtained by imitating NN policies. For each environment, we apply NeSyVerif to an ODT-policy trained with DTSemNet (Panda et al., 2026). NeSyVerif can also be applied to AADT-policies (Appendix E).

Implementation. We implemented NeSyVerif[2] in Python/PyTorch. The oblique abstractions are learned with the DTSemNet classifier using the counterexample-guided procedures of Sections 4 and 5, with training hyperparameters tuned using Optuna (Akiba et al., 2019) (details in Appendix F). The learned abstractions are then verified with the Gurobi LP solver (details in Appendix C). All experiments were run on Ubuntu 20.04.6 LTS with a 24-core Intel Xeon Gold 5220R CPU, 128 GB of RAM, and an NVIDIA GeForce RTX 3090 GPU (24 GB).

[1] https://github.com/VeriXAI/VeriTree

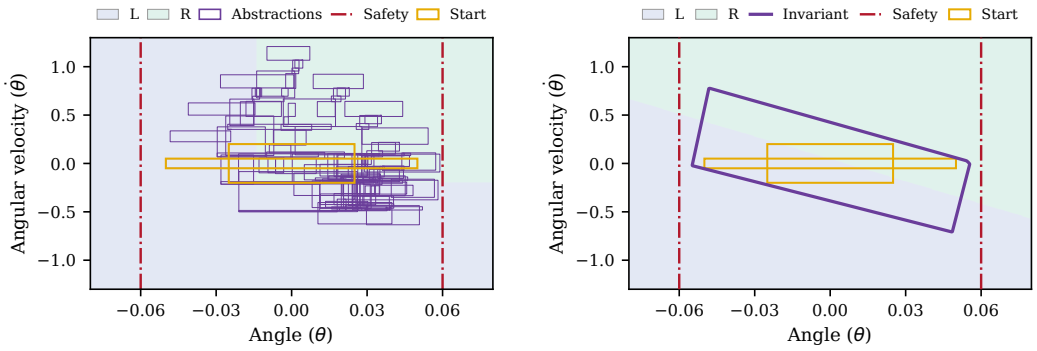
[2] Our anonymous source code: https://anonymous.4open.science/r/nesyverif



(a) AADT with VeriXAI

(b) ODT with NeSyVerif

Figure 3: Abstractions of the flowpipe for the quadrotor trajectories: ODT vs AADT.



(a) AADT-policy and flowpipe with VeriXAI

(b) ODT-policy and ODT-certificate with NeSyVerif

Figure 4: Safety verification for CartPole: AADT vs ODT.

Summary of results. Table 1 summarizes the results across all environments. First, the ODT-policies outperform the AADT-policies. On MountainCar, the ODT-policy reaches goal on average with 13% fewer steps (< 110 vs ≈ 126) using just one additional leaf; on CartPole, it gives a 23% higher reward with significantly lower variance and more compact ODT-policy; and on QuadRotor, it matches the AADT reward while being $\approx 10\times$ smaller (8 vs. 86 leaves). Second, NeSyVerif requires far fewer polytopes to verify the ODT-policies than VeriXAI needs for the AADT-policies. We observe reductions of roughly $11\times$ on MountainCar (129 vs. 1481), $98\times$ on QuadRotor (84 vs. 8241), and $68\times$ on CartPole (2 vs. 136). In all environments, the verified ODT-policies provably reach the goal while avoiding unsafe regions (MountainCar and QuadRotor) or remaining safe (CartPole).

In **MountainCar** an underpowered car must swing back and forth in a valley to reach the goal $x \geq 0.5$ without crossing $x = -1.25$; in **QuadRotor**, a quadrotor is tasked to follow a piecewise-linear reference trajectory for 30 steps without leaving the safety region. Due to the expressiveness of ODTs, the performance of ODT-policies is improved: in MountainCar, the ODT-policy has a larger safety margin of ≈ 0.21 to the unsafe boundary, whereas the AADT-policy margin is only ≈ 0.08. Similarly, on QuadRotor, the ODT-policy follows the reference closely, whereas the AADT-policies deviate significantly near $(x, y) = (0.8, 0.2)$ (see Fig. 3). Using oblique abstractions, NeSyVerif requires 1–2 orders of magnitude fewer polytopes; although the polytopes are loose in some instances (we optimized for fewer polytopes rather than tightness), they remain well within the safety boundary. NeSyVerif formally guarantees that the maximum number of steps to reach the goal is 129 on MountainCar and 30 on QuadRotor with the ODT-policies. Details are in Appendix E.

Safety (CartPole). In CartPole, the agent pushes a cart left or right to keep a pole upright; we consider its 2D version over (θ, ω) and verify the safety limit $|\theta| \leq 0.06$ rad, which is stricter than the environment limit of 0.21 rad. The two methods certify unbounded-horizon safety in fundamentally different ways. VeriXAI unrolls a flowpipe step by step and hopes it is eventually contracting, here found after 35 steps and 136 axis-aligned boxes (Fig. 4a). NeSyVerif instead learns one *ODT-certificate* (which exists by Theorem 2, here a polytope defined by 5 linear constraints (Fig. 4b), independent on the horizon, and which certifies every state inside it stays safe. See Appendix E.

7 Conclusion

In this paper, we addressed the problem of formally verifying CPS controlled by small efficient ODT-policies, a setting in which small & efficient AADT-policies are not known to exist. We proposed NeSyVerif, a *neurosymbolic* technique that *learns* polytope abstractions of the flowpipe and *symbolically verifies* they propagate through the policy and the system dynamics. The resulting over-approximations of the reachable states are verified to be sound. We also designed similar neurosymbolic approach to synthesize safety certificates under the form of ODT-certificates, which let us establish pure safety properties over an unbounded horizon in addition to reach-avoid properties. Our experiments on three CPS environments show that learned polytope abstractions can verify high-reward ODT-policies that are difficult to handle with grid-based or interval-based methods. In future work, we plan to extend our approach to higher-dimensional systems.

AI USE STATEMENT

In this work, we did not use generative AI tools for any of the tasks with required disclosure. In particular, we did not use them to develop the theoretical framework, formulate or prove mathematical claims, propose hypotheses, design the methodology or experiments, implement the proposed method, or interpret results. All core components of NeSyVerif and the experimental pipeline were designed and implemented by the authors. Generating synthetic datasets, cleaning or reformatting datasets, translation, and qualitative data analysis are not applicable to this work. Additionally, we used generative AI tools for tasks with recommended disclosure: (i) editing the manuscript to improve readability, including grammar correction and identifying notational inconsistencies; (ii) generating code for plots and figures; and (iii) writing small, non-critical code components, such as helper functions and patches, whose correctness does not affect the reported results. Generative AI tools were never used to write the full pipeline or any code implementing our core contributions. We have reviewed all AI-assisted work. All AI-suggested text edits were checked manually by the authors, and all AI-generated code was reviewed and tested by the authors before use, with figures verified against the underlying experimental data. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects, personally identifiable information, or sensitive data; all experiments are conducted on simulated control benchmark environments. We are not aware of any conflicts of interest or sponsorship concerns related to this work. The goal of this research is to improve the safety and trustworthiness of learned control policies by enabling their formal verification, which we view as a positive contribution to the safe deployment of AI systems. We note one potential risk: the verification guarantees provided by our approach hold only with respect to the specified system dynamics, abstractions, and safety properties. If these do not faithfully capture the real system, a verified policy could still behave unsafely in deployment. Practitioners should therefore validate modeling assumptions before relying on these guarantees in safety-critical settings.

REPRODUCIBILITY STATEMENT

We have made several efforts to ensure the reproducibility of our results. The complete source code for NeSyVerif, including policy training with DTSemNet, the verification pipeline, and scripts to reproduce all experiments and figures, is provided as supplementary material and is available at an anonymized repository: `https://anonymous.4open.science/r/nesyverif`. We will release the code publicly upon publication. The proposed method is described in Section 4 and 5, and all assumptions underlying our theoretical results are stated explicitly, with complete proofs provided in Appendix B. Details of the benchmark environments, system dynamics, safety specifications, hyperparameters, and computational resources are provided in Section 6 and Appendix F.

REFERENCES

Alessandro Abate, Daniele Ahmed, Alec Edwards, Mirco Giacobbe, and Andrea Peruffo. Fossil: a software tool for the formal synthesis of lyapunov functions and barrier certificates using neural networks. In *Proceedings of the 24th international conference on hybrid systems: computation and control*, pp. 1–11, 2021a.

Alessandro Abate, Daniele Ahmed, Mirco Giacobbe, and Andrea Peruffo. Formal synthesis of lyapunov neural networks. *IEEE Control. Syst. Lett.*, 5(3):773–778, 2021b. doi: 10.1109/LCSYS.2020.3005328. URL `https://doi.org/10.1109/LCSYS.2020.3005328`.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.

Matthias Althoff, Goran Frehse, and Antoine Girard. Set propagation techniques for reachability analysis. *Annual Review of Control, Robotics, and Autonomous Systems*, 4(1):369–395, 2021.

Guy Amir, Michael Schapira, and Guy Katz. Towards scalable verification of deep reinforcement learning. In *2021 formal methods in computer aided design (FMCAD)*, pp. 193–203. IEEE, 2021.

Edoardo Bacci and David Parker. Probabilistic guarantees for safe deep reinforcement learning. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pp. 231–248. Springer, 2020.

Edoardo Bacci, Mirco Giacobbe, and David Parker. Verifying reinforcement learning up to infinity. In *Proceedings of the International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, 2021.

Thom Badings, Wietze Koops, Sebastian Junges, and Nils Jansen. Policy verification in stochastic dynamical systems using logarithmic neural certificates. In *Computer Aided Verification - 37th International Conference, CAV 2025, Zagreb, Croatia, July 23-25, 2025, Proceedings, Part II*, pp. 349–375, 2025. doi: 10.1007/978-3-031-98679-6_16. URL https://doi.org/10.1007/978-3-031-98679-6_16.

Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. *Advances in neural information processing systems*, 31, 2018.

Sergiy Bogomolov, Goran Frehse, Mirco Giacobbe, and Thomas A Henzinger. Counterexample-guided refinement of template polyhedra. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 589–606. Springer, 2017.

Ya-Chien Chang, Nima Roohi, and Sicun Gao. Neural lyapunov control. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 3240–3249, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/2647c1dba23bc0e0f9cdf75339e120d2-Abstract.html.

Patrick Cousot. *Principles of abstract interpretation*. MIT Press, 2021.

Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Learning and verification of feedback control systems using feedforward neural networks. *IFAC-PapersOnLine*, 51(16): 151–156, 2018.

Tomer Eliyahu, Yafim Kazak, Guy Katz, and Michael Schapira. Verifying learning-augmented systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pp. 305–318, 2021.

Taha Entesari and Mahyar Fazlyab. Automated reachability analysis of neural network-controlled systems via adaptive polytopes. In *Learning for Dynamics and Control Conference*, pp. 407–419. PMLR, 2023.

Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In *International Conference on Computer Aided Verification*, pp. 379–395. Springer, 2011.

Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *HSCC*, volume 3414, pp. 291–305, 2005.

Didier Henrion and Andrea Garulli. *Positive polynomials in control*, volume 312. Springer Science & Business Media, 2005.

Chao Huang, Jiameng Fan, Xin Chen, Wenchao Li, and Qi Zhu. Polar: A polynomial arithmetic framework for verifying neural-network controlled systems. In *International Symposium on Automated Technology for Verification and Analysis*, pp. 414–430. Springer, 2022.

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*, pp. 169–178, 2019.

Zachary Jarvis-Wloszek, Ryan Feeley, Weehong Tan, Kunpeng Sun, and Andrew K. Packard. Some controls applications of sum of squares programming. In *42nd IEEE Conference on Decision and Control, CDC 2003, Maui, Hawaii, USA, December 9-12, 2003*, pp. 4676–4681, 2003. doi: 10.1109/CDC.2003.1272309. URL https://doi.org/10.1109/CDC.2003.1272309.

Ajaykrishna Karthikeyan, Naman Jain, Nagarajan Natarajan, and Prateek Jain. Learning accurate decision trees with bandit feedback via quantized gradient descent. *Transactions on Machine Learning Research*, 2022.

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*, pp. 97–117. Springer, 2017.

Sydney M Katz, Kyle D Julian, Christopher A Strong, and Mykel J Kochenderfer. Generating probabilistic safety guarantees for neural network controllers. *Machine Learning*, 112(8):2903–2931, 2023.

Hassan K Khalil and Jessy W Grizzle. *Nonlinear systems*, volume 3. Prentice hall Upper Saddle River, NJ, 2002.

Hector Kohler, Quentin Delfosse, Riad Akrour, Kristian Kersting, and Philippe Preux. Interpretable and editable programmatic tree policies for reinforcement learning. *arXiv preprint arXiv:2405.14956*, 2024.

Mathias Lechner, Dorde Zikelic, Krishnendu Chatterjee, and Thomas A. Henzinger. Stability verification in stochastic control systems via neural network supermartingales. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pp. 7326–7336, 2022. doi: 10.1609/AAAI.V36I7.20695. URL https://doi.org/10.1609/aaai.v36i7.20695.

Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In *International conference on runtime verification*, pp. 122–135. Springer, 2010.

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. *arXiv preprint arXiv:1509.02971*, 2015.

Ignacio D Lopez-Miguel, Ezio Bartocci, Thomas Eiter, and Martin Tappler. Orcaid: Oblique rule-based continuous-action interpretation for deep rl policies. *arXiv preprint arXiv:2607.07235*, 2026.

Garth P. McCormick. Computability of global solutions to factorable nonconvex programs: Part i—convex underestimating problems. *Mathematical Programming*, 10:147–175, 1976. doi: 10.1007/BF01580665.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Arnold Neumaier. The wrapping effect, ellipsoid arithmetic, stability and confidence regions. In *Validation Numerics: Theory and Applications*, pp. 175–190. Springer, 1993.

Grigory Neustroev, Mirco Giacobbe, and Anna Lukina. Neural continuous-time supermartingale certificates. In *Thirty-Ninth AAAI Conference on Artificial Intelligence, Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence, Fifteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2025, Philadelphia, PA, USA, February 25 - March 4, 2025*, pp. 27538–27546, 2025. doi: 10.1609/AAAI.V39I26.34966. URL https://doi.org/10.1609/aaai.v39i26.34966.

Subrat Prasad Panda, Blaise Genest, and Arvind Easwaran. Approximation-free differentiable oblique decision trees. *Journal of Machine Learning Research*, 27(125):1–31, 2026. URL http://jmlr.org/papers/v27/25-2047.html.

Pablo A Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. California Institute of Technology, 2000.

Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *International workshop on hybrid systems: Computation and control*, pp. 477–492. Springer, 2004.

Francesco Ranzato and Marco Zanella. Abstract interpretation of decision tree ensemble classifiers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 5478–5486, 2020.

Jiayang Ren, Qiangqiang Mao, Tianwei Zhao, and Yankai Cao. Exact learning of linear model predictive control laws using oblique decision trees with linear predictions. In *2025 IEEE 64th Conference on Decision and Control (CDC)*, pp. 7018–7023. IEEE, 2025.

Sriram Sankaranarayanan, Henny B Sipma, and Zohar Manna. Scalable analysis of linear systems using mathematical programming. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pp. 25–41. Springer, 2005.

Christian Schilling, Anna Lukina, Emir Demirović, and Kim Larsen. Safety verification of decision-tree policies in continuous time. *Advances in Neural Information Processing Systems*, 36:14750–14769, 2023.

Andrew Silva, Taylor Killian, Ivan Jimenez, Sung-Hyun Son, and Matthew Gombolay. Optimization methods for interpretable differentiable decision trees applied to reinforcement learning. In *International conference on artificial intelligence and statistics*, pp. 1855–1865. PMLR, 2020.

Gagandeep Singh, Markus Püschel, and Martin Vechev. A practical construction for decomposing numerical abstract domains. *Proceedings of the ACM on Programming Languages*, 2(POPL):1–28, 2017a.

Gagandeep Singh, Markus Püschel, and Martin Vechev. Fast polyhedra abstract domain. *SIGPLAN Not.*, 52(1):46–59, January 2017b. ISSN 0362-1340. doi: 10.1145/3093333.3009885. URL https://doi.org/10.1145/3093333.3009885.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, 2019.

Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T Johnson. Nnv: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International conference on computer aided verification*, pp. 3–17. Springer, 2020.

Hengjun Zhao, Xia Zeng, Taolue Chen, and Zhiming Liu. Synthesizing barrier certificates using neural networks. In *Proceedings of the 23rd international conference on hybrid systems: Computation and control*, pp. 1–11, 2020.

Dorde Zikelic, Mathias Lechner, Thomas A Henzinger, and Krishnendu Chatterjee. Learning control policies for stochastic systems with reach-avoid guarantees. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pp. 11926–11935, 2023a.

Dorde Zikelic, Mathias Lechner, Abhinav Verma, Krishnendu Chatterjee, and Thomas A. Henzinger. Compositional policy learning in stochastic control systems with formal guarantees. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023b. URL http://papers.nips.cc/paper_files/paper/2023/hash/95827e011b9e899f189a01fe2f4ef316-Abstract-Conference.html.

A NOTATION

Table 2 summarizes the notation used throughout the paper.

Table 2: Summary of notation.

Symbol	Meaning	Symbol	Meaning
System and specifications (Section 3)			
$S = (X, A, succ, I)$	control system	$X \subseteq \mathbb{R}^n$, x	state space, state
A, a	set of actions, action	$succ(x, a)$	successors of x under a
I	initial states	R, U	target, unsafe states
$\sigma : X \to A$	(ODT) policy	$succ_\sigma(x)$	$succ(x, \sigma(x))$
$Run^S(\sigma)$	runs under σ	$Reach^S(\sigma)$	states reachable under σ
S_δ, $succ_\delta$	δ-perturbed system	$Reach^S_\delta(\sigma)$, $\mathcal{R}_\delta$	states reachable in S_δ
$\|\|.\|\|$	L_∞-norm	$\mathcal{B}_\varepsilon(Z)$	open ε-ball around Z
K	Lipschitz constant	δ	perturbation radius
Oblique decision trees (Section 3)			
$\mathcal{T}$, $\mathcal{T}_x$	ODT, class output on x	$\mathcal{Y}$	classes ($\{0, 1\}$ or A)
V, V_k	internal nodes, a node	L, l	leaf nodes, a leaf
$W_i x + b_i \geq 0$	test at internal node i	$W_i \in \mathbb{R}^n$, $b_i \in \mathbb{R}$	weights, bias of node i
children, class	children map, leaf class	P_l, path(l)	polytope of leaf l, its path
Abstractions and certificates (Sections 4 and 5, Appendix C)			
$\mathcal{A}_i$	abstraction at step i	N	steps to reach R
B	ODT-certificate (invariant)	Q_j, a_j	policy leaf polytope, action
P_k, P_l	polytopes of $\mathcal{A}_{i-1}$, of $\mathcal{A}_i$	$W_{l,r}x + b_{l,r}$	r-th constraint of P_l
D	bounded box (verification)	η	MILP margin
Learning (Sections 4 and 5, Appendix F)			
C, C', C''	states, successors, candidates	Box	sampling box for C''
In, Un, Adm	inside, unsafe, admissible	C_2	boundary samples
In_2, Un_2, Adm_2	labels of boundary samples	$\mathcal{S}_c$	sampled successors of c
$p_1(c)$	probability that c is inside	c_{ce}	verifier counterexample
$\mathcal{L}_{in}$, $\mathcal{L}_{un}$, $\mathcal{L}_{adm}$	loss terms	$\mathcal{L}^{next}_{adm}$	contraction loss (safety)
$\mathcal{L}$, $\mathcal{L}_\alpha$	total loss	α, β	loss weights
$\bar{\alpha}$	threshold on α (Prop. 1)		
$\beta^{\max}$, $\beta^{\min}$	initial, final β	$\mathcal{H}$, κ	ladder of $\beta^{\max}$, factor
λ	determinant bound (In$_{Succ}$)	ρ	cube size (In$_{Succ}$)

B MISSING DETAILS AND PROOFS

B.1 PROOF OF PROPOSITION 1

Proposition 1 (restated). *Let $\mathcal{L}_\alpha = \mathcal{L}_{in} + \alpha(\mathcal{L}_{un} + \beta\mathcal{L}_{adm})$, with $\beta < 1$. There exists $\bar{\alpha} > 0$ such that for all $\alpha < \bar{\alpha}$, considering an abstraction $\mathcal{A}_\alpha$ minimizing the loss $\mathcal{L}_\alpha$, we have $In \subseteq \mathcal{A}_\alpha$.*

Proof. Let $\#_{adm}, \#_{un}$ be the number of states in Adm and Un respectively, and $\bar{\alpha} = \frac{1}{4(\#_{adm}+\#_{un})}$.

Assume by contradiction that there exists an $\alpha < \bar{\alpha}$ and $\mathcal{A}$ minimizing $\mathcal{L}_\alpha$ with a state $c \in In \setminus \mathcal{A}$. That is, $loss_{in}(c, \mathcal{A}) > 0.5$.

Consider any $\mathcal{A}'$ containing $\mathcal{A}$ and c with $loss_{in}(c, \mathcal{A}') < 0.25$. We have $\mathcal{L}_{in}(\mathcal{A}') = \sum_{c' \in In} loss_{in}(c', \mathcal{A}') < \mathcal{L}_{in}(\mathcal{A}) - 0.5 + 0.25 < \mathcal{L}_{in}(\mathcal{A}) - 0.25$, because $loss_{in}(c', \mathcal{A}') \leq loss_{in}(c', \mathcal{A})$ for all c' as $\mathcal{A} \subseteq \mathcal{A}'$. On the other hand, $\mathcal{L}_{adm}(\mathcal{A}') \leq \#_{adm}$ and $\mathcal{L}_{un}(\mathcal{A}') \leq \#_{un}$ because $loss_{un}(c', \mathcal{A}) \leq 1$ and $loss_{adm}(c', \mathcal{A}) \leq 1$ for all c'.

Thus, $\mathcal{L}_\alpha(\mathcal{A}') < \mathcal{L}_{in}(\mathcal{A}) - \frac{1}{4} + \bar{\alpha}(\#_{adm} + \#_{un}) \leq \mathcal{L}_\alpha(\mathcal{A})$, a contradiction with $\mathcal{A}$ minimizes $\mathcal{L}_\alpha$. □

B.2 Adaptive growth of the ODT Architecture

Gradient-based ODT learning (e.g. DTSemNet and DGT) typically learn a fixed ODT architecture. We thus design a heuristic to grow the ODT architectures to obtain tighter abstractions $\mathcal{A}_i$. The initial ODT architecture we consider corresponds to a unique polytope P with a reduced number of equations (e.g. $8\times$ number of dimensions). We then refine P, by first refining P to exclude admissible samples till improvement is negligeabe. Then we partition P in 2 polytopes using architecture in Fig. 1a, and proceed inductively on each, till improvement is negligeabe. More precisely:

1. We add equations / depth of the ODT if either:
   - Some states in Un are still in P: $Un \cap P \neq \emptyset$, or
   - if $Adm \cap P$ is too large (abstraction P too coarse).

   To do so, we remove all the samples not in P, and we learn a new P' from these samples. We append both sets of equations from P and P' to obtain the refined polytope P'' and repeat this step till the proportion of Adm in P is barely reduced by adding equations.
2. It could be that a convex solution (=unique Polytope) does not exist. We then add polytopes, partitioning the polytope P into 3 classes P_1 and P_2 and Out, using architecture in Fig. 1a. We stop this branch if $P_1 \sqcup P_2$ does not improve over P.
3. We proceed to refine inductively both P_1 and P_2 individually from step 1. This way, we can obtain as many polytopes with as many equations as necessary.

B.3 Proof of Proposition 2

Proof. Consider an arbitrary run $(x_i)_{i\geq 0}$ with $x_0 \in I = \mathcal{A}_0$. If $x_0 \in R$, the claim holds since $\mathcal{A}_0 \cap U = \emptyset$. Otherwise, we show by induction that, as long as the run has not reached R, $x_i \in \mathcal{A}_i$ and $x_i \notin U$ for $0 \leq i < N$. This holds for $i = 0$. For $i < N - 1$, the inductive inclusion gives $x_{i+1} \in \mathcal{A}_{i+1} \cup (R \setminus U)$. If $x_{i+1} \in R \setminus U$, the objective is satisfied. Otherwise, $x_{i+1} \in \mathcal{A}_{i+1}$, and $\mathcal{A}_{i+1} \cap U = \emptyset$ keeps the run safe. If the run has not reached R by step $N - 1$, the stopping condition gives $x_N \in R \setminus U$. Thus the run reaches R within N steps without entering U. Since the run was arbitrary, the result follows. □

B.4 Proof of theorem 2

Let σ be an ODT-policy for system $S = (X, A, succ, I)$, and let $(Q_j)_{j\in J}$ be the polytopes of the leaves of σ. Assume that σ is δ-robustly safe for $\delta > 0$. Recall that $\mathcal{R}_\varepsilon = Reach^S_\varepsilon(\sigma)$ for all $\varepsilon > 0$. It means that $\mathcal{R}_\delta \cap U = \emptyset$.

Assume the Lipschitz constants for $succ$ and σ (on each Q_j) are $\leq K/2$ for some $K \geq 2$, and consider $B^\star = \mathcal{B}_{\delta/K}(\mathcal{R}_{\delta/2})$. We show that $B^\star$ satisfies: $\mathcal{R}_{\delta/2} \subsetneq B^\star \subsetneq \mathcal{R}_\delta$ with:

- $I \subseteq B^\star$,
- $\mathcal{B}_{\delta/K}(B^\star) \cap U = \emptyset$, meaning that $B^\star$ is away from U,
- For all polytopes Q_j on which σ is Lipschitz continuous, for all $x \in Q_j \cap \mathcal{R}_{\delta/2}$ and $y \in Q_j$ with $||y - x|| < \delta/K$, $succ(y, \sigma(y)) \subseteq B^\star$.

It will be sufficient to obtain an ODT-certificate. Only the third item above is non trivial.

Proof. (of the third claim.)

First, since $x \in \mathcal{R}_{\delta/2}$ and $||y - x|| < \delta/K$, by assumption and definition, we have that $succ(y, \sigma(x)) \subseteq \mathcal{R}_{\delta/2}$.

Now, because both $succ$ and σ (on Q_j) are Lipschitz, we have that

$$\begin{aligned} succ(y, \sigma(y)) &\subseteq succ(y, \sigma(\mathcal{B}_{\delta/K}(x))) \\ &\subseteq succ(y, \mathcal{B}_{\delta/K}(\sigma(x))) \\ &\subseteq \mathcal{B}_{\delta/K}(succ(y, \sigma(x))) \\ &\subseteq \mathcal{B}_{\delta/K}(\mathcal{R}_{\delta/2}). \end{aligned}$$

□

To complete the proof of Theorem 2, we prove the following:

Lemma 1. *There exists a finite polytope decomposition $X = \bigcup_{j \in J} Q_j$, with J finite (the polytopes may include or exclude their boundaries), and Lipschitz continuous functions f_j assigned to each polytope Q_j that define σ:*

$$\sigma(x) = f_j(x) \quad \textit{if } x \in Q_j.$$

Proof. This is because σ is a piecewise Lipschitz continuous DT policy. □

Lemma 2. *There exists $\varepsilon > 0$ such that $\mathcal{R}_{\delta/2}$ is attractive in the region $\bigcup_{j \in J} \mathcal{B}_\varepsilon(\mathcal{R}_{\delta/2} \cap Q_j) \cap Q_j$.*

Proof. We use $\mathcal{R}^j$ to denote each $\mathcal{R}_{\delta/2} \cap Q_j$. By assumption, each Q_j is assigned a Lipschitz continuous function f_j with Lipschitz constant K_j. Let $K_\sigma = \max_{j \in J} K_j$ and $\varepsilon = \frac{\delta}{2K_\sigma + 2} < \delta/2$ (for the L_∞-norm; for other norms such as L_2 or L_1, ε should be smaller).

For a state y in $\mathcal{B}_\varepsilon(\mathcal{R}^j) \cap Q_j$, there exists $x \in \mathcal{R}^j \subseteq Q_j$ such that $||x - y|| < \varepsilon$. We consider $succ_\sigma(y)$ and $succ_\sigma(x)$.

Since $x, y \in Q_j$, $||\sigma(x) - \sigma(y)|| \leq K_j ||x - y|| < K_\sigma \varepsilon < \delta/2$. By assumptions, we have $succ_\sigma(y) \subseteq \mathcal{B}_{\delta/2}(succ_\sigma(x))$. Because $x \in \mathcal{R}^j \subseteq \mathcal{R}_{\delta/2}$, by definition, we have that $succ_\sigma(y) \subseteq \mathcal{R}_{\delta/2}$. □

This completes the proof of Theorem 2.

C VERIFICATION

All abstractions and certificates are checked by the same sound procedure, which we solve with Gurobi. Each abstraction $\mathcal{A}_i$ (and each ODT-certificate B) is an ODT over classes $\{0, 1\}$, so the set of states it classifies as inside is the union $\mathcal{A}_i = \bigcup_l P_l$ of the polytopes of its class-1 leaves l (Section 3), where $P_l = \{x \in X \mid W_{l,r} x + b_{l,r} \geq 0 \text{ for all } r\}$ and r ranges over the inequalities on the path to l. Likewise, the ODT policy σ partitions X into the polytopes Q_j of its leaves j, and each leaf plays a single action $a_j = \text{class}(j) \in A$. We prove

$$succ_\sigma(\mathcal{A}_{i-1}) \subseteq \mathcal{A}_i, \tag{1}$$

which implies the inductive condition $succ_\sigma(\mathcal{A}_{i-1}) \subseteq \mathcal{A}_i \cup (R \setminus U)$ of Section 4. The forward invariance of an ODT-certificate is the special case $\mathcal{A}_{i-1} = \mathcal{A}_i = B$ (Section 5). Because σ is constant on each Q_j, equation 1 splits into one query per non-empty pair (P_k, Q_j) with P_k a polytope of $\mathcal{A}_{i-1}$:

$$\exists\, x \in P_k \cap Q_j \cap D: \quad x' = succ(x, a_j), \qquad \forall l\ \exists r:\ W_{l,r} x' + b_{l,r} < 0, \tag{2}$$

where D is a bounded box and l ranges over the polytopes of $\mathcal{A}_i$. If every query is infeasible, then equation 1 holds.

Encoding. If $\mathcal{A}_i$ is a single polytope P_l, we solve one LP per row, $\min_x W_{l,r} x' + b_{l,r}$, and equation 1 holds if every minimum is ≥ 0. If $\mathcal{A}_i$ is a union, we add a binary $z_{l,r}$ per row with the constraints $W_{l,r} x' + b_{l,r} \leq -\eta + M_{l,r}(1 - z_{l,r})$ and $\sum_r z_{l,r} \geq 1$ for every l, with a margin $\eta \geq 0$ (possibly positive for counterexample search, and set to zero when proving containment), which gives a MILP. The constants $M_{l,r}$ are computed from D, so the encoding is sound.

Nonlinear dynamics. When $succ(\cdot, a)$ is nonlinear, we replace it by a linear relaxation that contains its graph over the current variable bounds. We use secant and tangent bounds for $\sin$, $\cos$, squares

and reciprocals (secants as upper bounds and tangents as lower bounds on convex intervals, and vice versa on concave intervals), and McCormick envelopes (McCormick, 1976) for bilinear terms. Infeasibility of all violation queries in the certification mode (with $\eta = 0$ for the MILP) therefore proves containment. A feasible witness x is re-simulated with the exact dynamics: if its successor leaves $\mathcal{A}_i$, it is a real counterexample c_{ce} and is returned to training (Algorithms 1 and 2); otherwise the query is inconclusive. Table 3 lists the instantiation for each environment.

Table 3: Verification setup for each environment.

	MountainCar	**QuadRotor**	**CartPole**
Property	$succ_\sigma(\mathcal{A}_{i-1}) \subseteq \mathcal{A}_i$	$succ_\sigma(\mathcal{A}_{i-1}) \subseteq \mathcal{A}_i$	$succ_\sigma(B) \subseteq B$
Policy pieces Q_j	one action per step (open loop)	8 leaves	2 leaves
Target set	single polytope	union of polytopes	single polytope
Dynamics	semi-implicit Euler	affine for each action	RK4 ($\tau = 0.02$)
Nonlinearity	$\cos(3x)$, 100+ cells	none (exact)	relaxed at each RK4 stage
Encoding	LP per row and cell	MILP, $\eta = 0$	LP per row

D Environment and ODT Policy Details

All benchmarks follow Schilling et al. (2023). For each environment, we give the system $S = (X, A, succ, I)$, the verification objective, and the ODT policy σ learnt with DTSemNet. We use the specifications of Section 3. A policy σ satisfies the reach-avoid specification (R, U) within N steps if every run $(x_i)_{i\in\mathbb{N}} \in Run^S(\sigma)$ satisfies

$$\exists\, k \le N: \quad x_k \in R \quad \text{and} \quad x_i \notin U \ \text{ for all } i \le k. \tag{3}$$

It satisfies the safety specification U, i.e. $Reach^S(\sigma) \cap U = \emptyset$, if every such run satisfies

$$x_k \notin U \quad \text{for all } k \in \mathbb{N}. \tag{4}$$

D.1 MountainCar (Reach-avoid)

Environment. A car in a sinusoidal valley must build momentum to reach the top of the right hill. The state is (x, v) (position, velocity), the action set is $A = \{0, 1, 2\}$ (left, none, right), and $succ$ is given by

$$v_{k+1} = v_k + (a-1)\,0.001 - 0.0025\cos(3x_k), \qquad x_{k+1} = x_k + v_{k+1}.$$

Verification objective. Reach-avoid equation 3 with

$$I = [-0.505, -0.445]\times[0, 0.0001],\ R = \{(x, v) \mid x \ge 0.5\},\ U = \{(x, v) \mid x < -1.25\},\ N = 200.$$

Policy. The 3-leaf ODT policy is given in Policy MountainCar.

Policy: MountainCar 3-leaf ODT policy σ learnt with DTSemNet.

Require: (x, v)
1: **if** $10\,x + 100\,v \le 0$ **then**
2: **if** $5\,x - 110\,v + 2 \le 0$ **then return** $a = 2$ ▷ right
3: **else return** $a = 0$ ▷ left
4: **end if**
5: **else return** $a = 2$ ▷ right
6: **end if**

D.2 QuadRotor (Reach-avoid)

Environment. A quadrotor must track a piecewise-linear reference trajectory using bang-bang control. The state is (x, y, z, v_x, v_y, v_z), where $q = (x, y, z)$ is the position relative to the reference

trajectory and $v = (v_x, v_y, v_z)$ the absolute velocity. The control is discretized into 8 actions: each action $a \in A = \{0, \dots, 7\}$ fixes pitch $\vartheta \in \{\pm 0.1\}$, roll $\varphi \in \{\pm 0.1\}$, and thrust $T \in \{11.81, 7.81\}$, in the order

$$(\vartheta, \varphi, T)_a \in \big\{(+,+,\mathrm{hi}), (+,+,\mathrm{lo}), (+,-,\mathrm{hi}), (+,-,\mathrm{lo}), (-,+,\mathrm{hi}), (-,+,\mathrm{lo}), (-,-,\mathrm{hi}), (-,-,\mathrm{lo})\big\},$$

for $a = 0, \dots, 7$, where $\pm$ stands for ± 0.1, $\mathrm{hi} = 11.81$, and $\mathrm{lo} = 7.81$. With acceleration $u_a = (g \tan\vartheta, -g \tan\varphi, T - g)$, $g = 9.81$, and control period $\tau = 0.2$, the dynamics are

$$q_{k+1} = q_k + (v_k - b_k)\,\tau + \tfrac{1}{2}\,u_a\,\tau^2, \qquad v_{k+1} = v_k + u_a\,\tau,$$

which is affine in the state for each action. The reference velocity b_k is piecewise constant: $b_k = (0.25, -0.25, 0)$ for $k < 10$, $(0.25, 0.25, 0)$ for $10 \le k < 20$, $(0, 0.25, 0)$ for $20 \le k < 25$, and $(-0.25, 0.25, 0)$ for $k \ge 25$. The reference trajectory therefore passes through the waypoints $(0, 0)$, $(0.5, -0.5)$, $(1, 0)$, $(1, 0.25)$, and $(0.75, 0.5)$ in the (x, y)-plane, and we write r_k for its position at step k.

Verification objective. Reach-avoid equation 3 with $N = 30$, where the goal is checked at the final step:

$$x_k \notin U \ \text{ for all } k \le 30 \quad \text{and} \quad x_{30} \in R, \qquad I = [-0.05, 0.05]^2 \times \{0\}^4.$$

Both sets are defined on the absolute position $(x, y) + r_k$. The goal R is the disc of radius 0.26 centred at $(0.81, 0.5)$, intersected with $\{y \le 0.5\}$. The unsafe set U is the complement of a corridor of half-width 0.32 around the reference trajectory, closed at both ends with a margin of 0.15.

Policy. The 8-leaf ODT policy is given in Policy QuadRotor. Each leaf plays a different action, so all 8 actions are used.

Policy: QuadRotor 8-leaf ODT policy σ learnt with DTSemNet. The comment next to each action gives (ϑ, φ, T).

Require: state (x, y, z, v_x, v_y, v_z)

$h_1 \leftarrow 94.22\,x + 0.44\,y + 0.85\,z + 30.78\,v_x + 0.63\,v_y - 0.19\,v_z - 6.24$
$h_2 \leftarrow 5.18\,x - 123.26\,y + 10.80\,z + 0.85\,v_x - 37.37\,v_y - 0.01\,v_z - 3.80$
$h_3 \leftarrow 14.07\,x - 5.75\,y + 141.26\,z + 10.65\,v_x - 5.58\,v_y + 15.34\,v_z - 6.01$
$h_4 \leftarrow -4.62\,x + 14.41\,y + 135.66\,z + 7.53\,v_x + 9.55\,v_y + 15.75\,v_z - 3.99$
$h_5 \leftarrow -0.82\,x - 95.49\,y + 5.10\,z - 1.15\,v_x - 31.96\,v_y - 1.02\,v_z - 2.31$
$h_6 \leftarrow -4.86\,x + 2.03\,y + 160.82\,z - 2.30\,v_x + 0.53\,v_y + 24.21\,v_z - 8.20$
$h_7 \leftarrow 8.25\,x + 22.66\,y + 138.41\,z - 1.26\,v_x + 10.73\,v_y + 17.44\,v_z - 4.30$
if $h_1 \le 0$ **then**
  if $h_2 \le 0$ **then**
    if $h_3 \le 0$ **then return** $a = 0$ ▷ $(+0.1, +0.1, 11.81)$
    else return $a = 1$ ▷ $(+0.1, +0.1, 7.81)$
    end if
  else
    if $h_4 \le 0$ **then return** $a = 2$ ▷ $(+0.1, -0.1, 11.81)$
    else return $a = 3$ ▷ $(+0.1, -0.1, 7.81)$
    end if
  end if
else
  if $h_5 \le 0$ **then**
    if $h_6 \le 0$ **then return** $a = 4$ ▷ $(-0.1, +0.1, 11.81)$
    else return $a = 5$ ▷ $(-0.1, +0.1, 7.81)$
    end if
  else
    if $h_7 \le 0$ **then return** $a = 6$ ▷ $(-0.1, -0.1, 11.81)$
    else return $a = 7$ ▷ $(-0.1, -0.1, 7.81)$
    end if
  end if
end if

D.3 CartPole (Safety)

Environment. A pole is hinged on a cart that is pushed left or right to keep the pole upright. The state is (p, v, θ, ω) (cart position and velocity, pole angle and angular velocity), the action set is $A = \{-1, 1\}$, and

$$\dot{p} = v, \quad \dot{v} = \psi - \tfrac{1}{22}\,\phi\cos\theta, \quad \dot{\theta} = \omega, \quad \dot{\omega} = \phi, \qquad \phi = \frac{9.8\sin\theta - \psi\cos\theta}{2/3 + \frac{5}{11}\cos^2\theta}, \quad \psi = \frac{10a + 0.05\,\omega^2\sin\theta}{1.1},$$

with a control period of 0.02, integrated with RK4 to obtain $succ$. The environment terminates when $|\theta| > 0.21$ rad (12°). The dynamics of (θ, ω) do not depend on (p, v), so we use the 2D CartPole over (θ, ω).

Verification objective. Safety equation 4 over an unbounded horizon, with the enhanced safety limit 0.06 rad, which is stricter than the environment limit:

$$I = [-0.05, 0.05]^4 \cup \big(\{0\}^2 \times [-0.025, 0.025] \times [-0.2, 0.2]\big), \qquad U = \{|\theta| > 0.06\}.$$

It suffices to find an ODT-certificate B with $I \subseteq B$, $B \cap U = \emptyset$, and $succ_\sigma(B) \subseteq B$ (Section 5).

Policy. The 2-leaf ODT policy depends only on (θ, ω) and is given in Policy CartPole.

Policy: CartPole 2-leaf ODT policy σ learnt with DTSemNet.

Require: (θ, ω)
1: **if** $19.90\,\theta + 2.58\,\omega - 0.10 \le 0$ **then return** $a = -1$ ▷ left
2: **else return** $a = +1$ ▷ right
3: **end if**

E Additional Results and Discussion

Here we provide additional environment-specific results such as verified bounds: the maximal number of steps before reaching goal in MountainCar, and the maximal pole angle for CartPole.

E.1 MountainCar (Reach-avoid)

Policy performance. Table 4 reports the results. We first compare the 2-leaf AADT policy of Schilling et al. (2023) (Fig. 6a) with the 3-leaf ODT policy that we learn directly with DTSemNet (Fig. 6b). The AADT policy needs 126 steps on average to reach the goal, and its trajectories come close to the unsafe boundary $x = -1.25$. The ODT policy needs about 110 steps on average, and its trajectories stay far (0.21) from this boundary, whereas, AADT policy reaches very close (0.08) to safety boundary. A neural network policy trained with deep RL is only marginally more efficient than the ODT policy (reward -109.0 vs. -109.9, a difference below 1%), but it cannot be verified with existing tools.

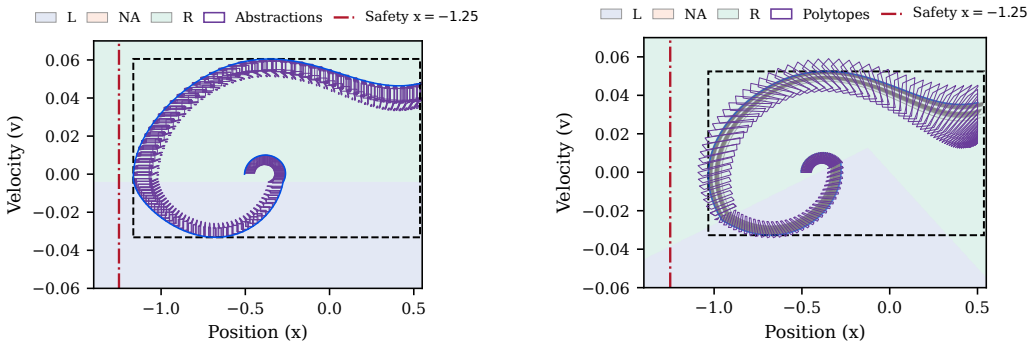
Verification. VeriXAI can only verify the less efficient AADT policy (reward -126.3 vs. -109.9 for the ODT policy). It needs 1481 rectangles (Fig. 5) to over-approximate the trajectories and guarantees that the goal is reached within 132 steps. In contrast, NeSyVerif can verify both AADT and ODT policies. For the AADT policy, it needs only 153 polytopes (Fig. 6a), one per step, each defined by 4 linear constraints, and guarantees that the goal is reached within 153 steps. NeSyVerif thus reduces the number of polytopes by almost an order of magnitude compared to VeriXAI, at the cost of a slightly looser step bound. For the ODT policy, it needs 129 polytopes (Fig. 6b) and guarantees that the goal is reached within 129 steps, the best result in both average and max steps.

E.2 QuadRotor (Reach-avoid)

Policy performance. Table 1 reports the results, and Fig. 3 shows the abstractions of the trajectories. The AADT policy of Schilling et al. (2023) and our ODT policy have similar performance, since the objective is to follow the reference trajectory. However, the ODT policy is an order of magnitude smaller, with 8 leaves instead of 86, which makes it considerably easier to obtain reachable

Table 4: Rewards for AADT, ODT and NN policies on MountainCar. The policies are evaluated over 1000 episodes. For VeriXAI and NeSyVerif, we report the number N_{poly} of polytopes generated to make the proof, as well as a formal upper bound maximum steps (Max Steps) on the number of steps to reach R.

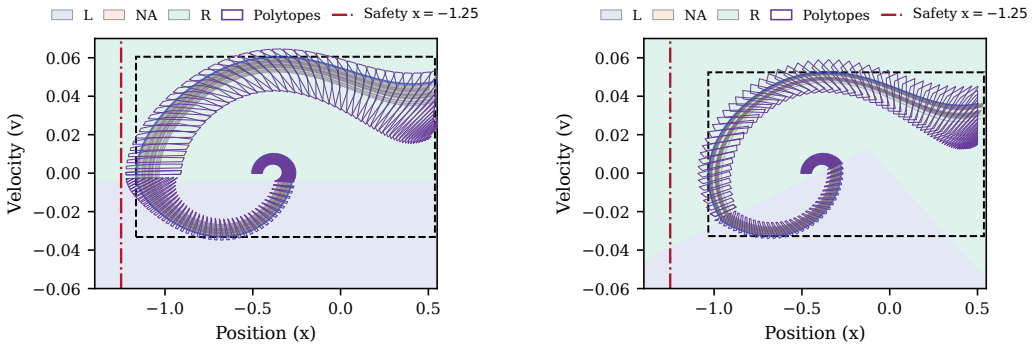
MountainCar	Reward	N_{poly}	Max Steps
VeriXAI (AADT)	-126.3	1481	132
NeSyVerif (AADT)	-126.3	153	153
NeSyVerif (ODT)	-109.9	**129**	**129**
NeuralNetwork	-109.0	N/A	N/A



(a) AADT with VeriXAI

(b) ODT with NeSyVerif

Figure 5: Comparision between VeriTree and NeSyVerif.



(a) AADT with NeSyVerif

(b) ODT with NeSyVerif

Figure 6: NeSyVerif used to obtain abstractions for both AADT and ODT policies. Abstractions $\mathcal{A}_i$ computed by NeSyVerif for MountainCar: (a) the 2-leaf AADT policy produced by imitation, and (b) the 3-leaf ODT policy learnt by DTSemNet.

abstractions. It is observed that the ODT policy follows the reference closely, whereas the AADT abstraction contains axis-aligned boxes that deviate significantly near $(x, y) = (0.8, 0.2)$.

Verification. VeriXAI verifies the AADT policy using 8241 axis-aligned boxes (Fig. 3a). In contrast, NeSyVerif verifies the ODT policy using only 84 polytopes (Fig. 3b), a reduction of about two orders of magnitude. The resulting abstraction guarantees that the goal is reached within 30 steps without entering the unsafe region.

E.3 CartPole (Safety)

We now turn to CartPole, with a safety goal. We use its 2D version with state (θ, ω), respectively the angle to vertical and the angular velocity of the pole, and 2 discrete actions (push left, push right). The environment limit on the pole angle is ± 0.21 rad (12°) from the vertical position. For verification, we use an enhanced safety limit $|\theta| \leq 0.06$ rad, which is stricter than the environment limit. Table 5 reports the results.

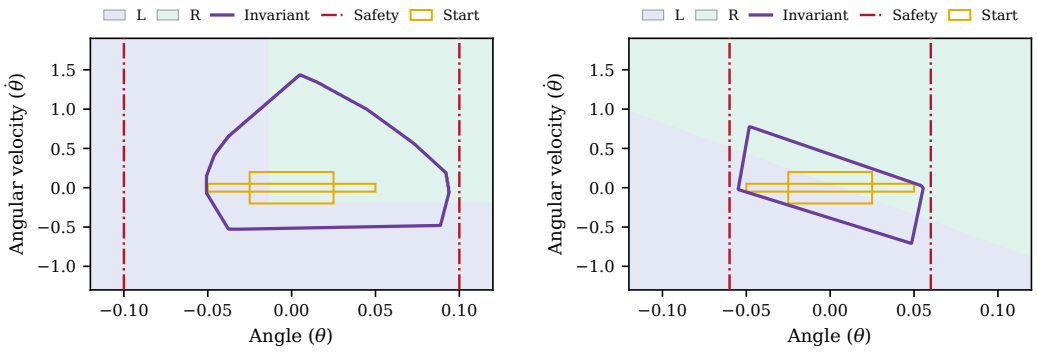
Policy performance. We consider an AADT policy from Schilling et al. (2023). It is the first policy for CartPole for which there is a proof of safety for the pole. The ODT policy learnt with DTSemNet has only 2 leaves, and it is also more efficient (reward 499.8 vs. 407.5).

Verification. Here our aim is to show that we can learn an ODT-certificate for the policy, whereas the original proof of safety was building inductively on the number of steps i abstractions for reachable sets after i steps from the initial configurations, and hoping that a fix point would be eventually reached. This is the case after 35 steps, and 136 polytopes have been built (see Fig. 4a). By comparison, using NeSyVerif, we learnt a unique polytope with 4 linear constraints as the first candidate ODT-certificate B. This first candidate was actually not forward invariant. We fed the learner with the counterexample, and the second refinement iteration learnt a second polytope with 1 linear constraint, grafted onto the first one, giving a final region defined by 5 linear constraints. This second candidate was then verified as a valid ODT-certificate (Fig. 4b in Section 6). It contains the initial set and all sampled reachable states, and it keeps the pole angle within the enhanced safety limit $|\theta| \leq 0.06$ rad, so it certifies safety for an unbounded horizon.

We also verify an AADT policy using NeSyVerif to demonstrate its application to axis-aligned policies, as illustrated in Fig. 7. Since the policy does not perform well, its certificate requires more equations (12, compared with 6 for the ODT certificate) and more iterations to converge. The invariant certificate for the AADT policy also has a looser safety threshold of 0.10. Although this is still below the environment's safety limit, it shows that the ODT policy is not only more performant but also safer.

Table 5: Average rewards for CartPole policies evaluated over 1000 episodes. For VeriXAI and NeSyVerif, we report the number N_{poly} of polytopes generated to make the proof, as well as a formal upper bound $Bound$ on the maximal deviation of the pole (in radians) to the vertical.

CartPole	Reward	N_{poly}	Bound (rad)
VeriXAI (AADT)	407	136	± 0.06
NeSyVerif (AADT)	407	6	± 0.10
NeSyVerif (ODT)	499	2	± 0.06



(a) ODT-certificate for AADT-policy

(b) ODT-certificate for ODT-policy

Figure 7: ODT-certificate with NeSyVeri for AADT-policy and ODT-policy.

Invariance of the ODT-certificate. Fig. 8 illustrates why the learnt polytope is forward invariant. We sample states in a thin band just inside its boundary, and move them one step forward under the closed-loop dynamics $succ_\sigma$. The successors all move further inside the polytope, so no state near

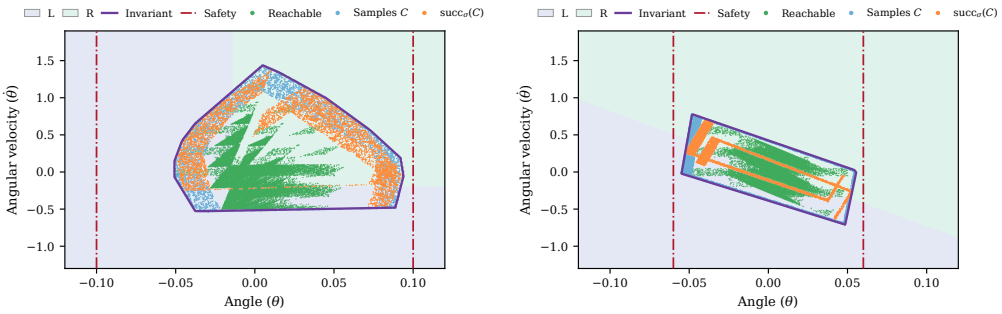


Figure 8: Illustration of the forward invariance of the ODT-certificate B for the CartPole AADT-policy (left) and ODT-policy (right). States are sampled just inside (blue) the boundary of B and forward stepped under $succ_\sigma$. None cross the boundary, consistent with $succ_\sigma(B) \subseteq B$.

the boundary escapes it, which is the invariance condition $succ_\sigma(B) \subseteq B$. This figure is only an illustration and sanity check using sampling: the formal guarantee comes from the LP verification of Appendix C.

F Training Details

We learn each abstraction $\mathcal{A}_i$ (Section 4) and each ODT-certificate B (Section 5) by supervised binary classification over classes $\mathcal{Y} = \{0, 1\}$: a state is labelled 1 if it should lie inside the set and 0 if it should lie outside. The set is represented by an ODT, which we train with the weighted losses of Sections 4 and 5. Below, we first describe how we generate the training data and then give the training hyperparameters. The source code for all experiments is available at `https://anonymous.4open.science/r/nesyverif`.

F.1 Sampling and Dataset

To sample states near the decision boundary, we use rejection sampling. We first draw states uniformly from a bounding box that contains the current set $\mathcal{A}_i$ or B, such as the bounding box of Adm. Each state c is then passed through DTSemNet, the neural representation of the ODT, which outputs the probability $p_1(c)$ that c lies inside the set; states with $p_1(c) > 0.5$ are classified as inside (class 1) and the others as outside (class 0). We accept a state only if $p_1(c) \in [p_{\text{low}}, p_{\text{high}}]$, with $p_{\text{low}} = 0.4$ and $p_{\text{high}} = 0.7$ in our experiments, which yields samples on both sides of the boundary. The width of this interval controls how close the accepted samples are to the boundary.

F.2 Training Setup and Hyperparameter

Every abstraction $\mathcal{A}_i$ and ODT-certificate B is a DTSemNet binary classifier, and its class-1 leaves form a union of polytopes. We train $\mathcal{A}_i$ with the loss of Section 4, $\mathcal{L} = \mathcal{L}_{in} + \alpha\,(\mathcal{L}_{un} + \beta\,\mathcal{L}_{adm})$ with $\alpha = 1$, and B with the loss of Section 5, which additionally contains $\mathcal{L}_{adm}^{next}$ at the level of $\mathcal{L}_{un}$. The weight β trades tightness against soundness. It starts at $\beta^{\max}$, which favours a tight set, and decays to $\beta^{\min}$ during training, so that classifying In correctly takes priority. A run that still misclassifies some $c \in In$ is restarted with the next, smaller $\beta^{\max}$ from a decreasing ladder $\mathcal{H}$. For QuadRotor and CartPole, we set $\beta^{\max} = \kappa\,|Adm|/|In|$ to correct for class imbalance, and κ runs through the ladder. We use AdamW with a cosine learning-rate schedule (with warm restarts for QuadRotor and CartPole) and gradient clipping. We tune hyperparameters with TPE in Optuna (Akiba et al., 2019) for MountainCar and with random search (10 trials of 50 epochs) for the other two environments. Table 6 summarises the settings.

F.3 Reach-avoid Abstractions (MountainCar and QuadRotor)

Algorithm 1 gives the pseudocode of the NeSyVerif pipeline of Fig. 2, which builds $\mathcal{A}_1, \mathcal{A}_2, \ldots$ step by step. *Build data.* At step i, we sample states C in $\mathcal{A}_{i-1}$ and their successors $C' = succ_\sigma(C)$,

Table 6: Training settings. Sets written with braces are search spaces, and single values are fixed.

	MountainCar	**QuadRotor**	**CartPole**
Hyperparameter search	TPE, once per policy	random, per step and group	random, per iteration
Tree height	$\{4, 6\}$	$\{6, 8, 10, 12\}$	$\{6, 8, 12\}$
Batch size	8192	$\{1024, 2048, 4096\}$	$\{1024, 2048, 4096\}$
Learning rate	10^{-2}	$\log \mathcal{U}[10^{-3}, 10^{-1}]$	$\log \mathcal{U}[10^{-3}, 10^{-1}]$
Max. epochs	4000	500	500
Ladder $\mathcal{H}$	$\beta^{\max} = \beta_0(1 - j/5)$, $\beta_0 \in \{1, 2\}$	$\kappa \in \{0.2, 0.1, 0.01, 0.001\}$	$\kappa \in \{0.5, 0.2, 0.1, 0.01\}$
Floor $\beta^{\min}$	10^{-3}	$10^{-3}\beta^{\max}$	$10^{-4}\beta^{\max}$
$\lvert In \rvert$	10^4 per step	$\geq 2 \times 10^4$ per group	8.2×10^4
Outer loop	129 steps	30 steps (84 group models)	2 iterations

which are inside by construction. Candidates C'' are sampled in a bounding box around C'. Each $c'' \in C''$ is tagged *inside* if the symbolic predecessor check In_{Succ} certifies a predecessor in $\mathcal{A}_{i-1}$, and *unsafe* if a closed-loop rollout from c'' violates (R, U); the remaining states are *admissible* (LABEL in Algorithm 1, see Section 4). *Learn.* We train $\mathcal{A}_i$, lowering the weight of $\mathcal{L}_{un}, \mathcal{L}_{adm}$ along the ladder $\mathcal{H}$ until $In \subseteq \mathcal{A}_i$. *Falsify.* We sample new states near the boundary of $\mathcal{A}_i$ and label them in the same way, giving In_2, Un_2, Adm_2. *Missed successors* ($In_2 \not\subseteq \mathcal{A}_i$) and *unsafe states inside* ($Un_2 \cap \mathcal{A}_i \neq \emptyset$) are counterexamples: the new samples are added to the training data and $\mathcal{A}_i$ is retrained. *Verify.* Once no counterexample remains, we verify $succ_\sigma(\mathcal{A}_{i-1}) \subseteq \mathcal{A}_i \cup (R \setminus U)$ formally (Appendix C); a counterexample returned by the verifier is added to Un and $\mathcal{A}_i$ is retrained.

MountainCar. Each step learns a single polytope. The 129 steps cover the five phases of the policy (right, transit, left, transit, right). In the transit phases, the polytope is also constrained by the switching hyperplane of policy σ. *QuadRotor.* At a fixed step, (z, v_x, v_y, v_z) take only 1–4 distinct values. We group In by these values and learn one polytope over (x, y) per group, which gives 84 polytopes over 30 steps.

Algorithm 1 Counterexample-guided learning of reach-avoid abstractions.

Require: policy σ, successor function $succ$, initial set I, specification (R, U), ladder $\mathcal{H}$

$\mathcal{A}_0 \leftarrow I$
for $i = 1, 2, \ldots$ until $succ_\sigma(\mathcal{A}_{i-1}) \subseteq R \setminus U$ **do**
  $C \leftarrow \text{SAMPLE}(\mathcal{A}_{i-1})$; $C' \leftarrow succ_\sigma(C)$; $C'' \leftarrow \text{SAMPLEBOX}(C')$ ▷ 1. build data
  $In, Un, Adm \leftarrow \text{LABEL}(C'', \mathcal{A}_{i-1})$; $In \leftarrow In \cup C'$ ▷ In_{Succ} + rollout
  if $In \cap Un \neq \emptyset$ **then** backtrack
  loop
    for $\beta^{\max} \in \mathcal{H}$ **do** ▷ 2. learn
      $\mathcal{A}_i \leftarrow \text{TRAIN}(In, Un, Adm;\ \beta^{\max})$; **if** $In \subseteq \mathcal{A}_i$ **then break**
    end for
    $In_2, Un_2, Adm_2 \leftarrow \text{LABEL}(\text{SAMPLEBOUNDARY}(\mathcal{A}_i), \mathcal{A}_{i-1})$ ▷ 3. falsify
    $In \leftarrow In \cup In_2$; $Un \leftarrow Un \cup Un_2$; $Adm \leftarrow Adm \cup Adm_2$
    if $In_2 \not\subseteq \mathcal{A}_i$ **or** $Un_2 \cap \mathcal{A}_i \neq \emptyset$ **then continue** ▷ retrain
    $(ok, c_{ce}) \leftarrow \text{VERIFY}\big(succ_\sigma(\mathcal{A}_{i-1}) \subseteq \mathcal{A}_i \cup (R \setminus U)\big)$ ▷ 4. verify
    if ok **then break** ▷ return $\mathcal{A}_i$, next step $i{+}1$
    $Un \leftarrow Un \cup \{c_{ce}\}$ ▷ retrain
  end loop
end for

F.4 ODT-CERTIFICATE (CARTPOLE)

Algorithm 2 learns the ODT-certificate B of Section 5 over (θ, ω). It follows the pipeline of Fig. 2 with $\mathcal{A}_{i-1} = \mathcal{A}_i = B$, where the predecessor check is replaced by a successor check. The first candidate is fitted tightly around the sampled δ-perturbed reachable set $\mathcal{R}_\delta$ (which plays the role of In), with $U \cup Un$ outside. Each refinement iteration samples 2×10^4 states near the boundary of B and, for each sampled $c \in B$, a set $\mathcal{S}_c$ of successors in $succ_\sigma(c)$. States with $\mathcal{S}_c \not\subseteq B$ are counterexamples and are added to Un, and the others to Adm. It then trains a subtree on the refined

data and grafts it onto the class-1 leaf of B. An LP removes redundant constraints. The refinement converged after 2 iterations: the first iteration learns a polytope with 4 linear constraints, and the second iteration learns a polytope with 1 linear constraint, which is grafted onto the first, giving the final ODT-certificate defined by 5 linear constraints. In Tables 1 and 5, $N_{poly} = 2$ counts these 2 polytopes learnt over the 2 iterations.

Algorithm 2 Counterexample-guided learning of the ODT-certificate.

Require: policy σ, successor function $succ$, unsafe set U, perturbation δ, ladder $\mathcal{H}$

$\mathcal{R}_\delta \leftarrow \textsc{Sample}(Reach^S_\delta(\sigma))$; $Adm \leftarrow \textsc{SampleAround}(\mathcal{R}_\delta)$; $Un \leftarrow \emptyset$
$B \leftarrow \textsc{Train}(\mathcal{R}_\delta, Un \cup U, Adm;\ \beta^{\max} \in \mathcal{H})$
loop
  $C_2 \leftarrow \textsc{SampleBoundary}(B)$; $\mathcal{S}_c \leftarrow \textsc{Sample}(succ_\sigma(c))$ for each $c \in C_2 \cap B$ ▷ falsify
  $Un_2 \leftarrow \{c \in C_2 \cap B : \mathcal{S}_c \not\subseteq B\}$; $Adm \leftarrow Adm \cup (C_2 \setminus Un_2)$
  if $Un_2 = \emptyset$ **then** ▷ verify
    $(ok, c_{ce}) \leftarrow \textsc{Verify}(succ_\sigma(B) \subseteq B)$; **if** ok **then return** B
    $Un_2 \leftarrow \{c_{ce}\}$
  end if
  $Un \leftarrow Un \cup Un_2$ ▷ counterexamples
  $B' \leftarrow \textsc{Train}(\textsc{Refine}(B, Un);\ \beta^{\max} \in \mathcal{H})$; $B \leftarrow \textsc{Reduce}(\textsc{Graft}(B, B'))$
end loop