Safety Verification of Tree-Ensemble Policies via Predicate Abstraction

Chaahat Jain[1], **Lorenzo Cascioli**[2], **Laurens Devos**[2], **Marcel Vinzent**[1], **Marcel Steinmetz**[3], **Jesse Davis**[2], **Jörg Hoffmann**[1,4]

[1] Saarland University, Saarland Informatics Campus, Germany
[2] Department of Computer Science, KU Leuven, Leuven, Belgium
[3] LAAS-CNRS, ANITI, Université de Toulouse, INSA, France
[4] German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany
⟨last⟩@cs.uni-saarland.de, ⟨first⟩.⟨last⟩@kuleuven.be, marcel.steinmetz@laas.fr

Abstract

Learned action policies are gaining traction in AI, but come without built-in safety guarantees. Recent work devised a method for safety verification of neural policies via predicate abstraction. Here we extend this approach to policies represented by tree ensembles, through replacing the underlying SMT queries with queries that can be dispatched by Veritas, a reasoning tool dedicated to tree ensembles. The query language supported by Veritas is limited, and we show how to encode richer constraints we need into additional trees added to the policy ensemble. On benchmarks previously used to evaluate neural policy verification, we find that (1) tree ensembles can learn policies highly competitive with the neural policies, while at the same time (2) these tree-ensemble policies can be orders of magnitude faster to verify.

1 Introduction

Learned action policies are gaining traction in AI (e.g., Mnih et al. 2015; Silver et al. 2016, 2018), including in AI planning (Issakkimuthu, Fern, and Tadepalli 2018; Groshev et al. 2018; Garg, Bajpai et al. 2019; Toyer et al. 2020; Karia and Srivastava 2021; Ståhlberg, Bonet, and Geffner 2022a,b). However, such policies come without built-in safety guarantees. Given a policy π, a start condition ϕ_0, and an unsafety condition ϕ_u, how to verify whether an unsafe state $s^u \models \phi_u$ is reachable from a given start state $s^0 \models \phi_0$ under π?

Research on this question is still in its early stages. In the formal methods community, prominent lines of works address neural controllers of dynamical systems (Sun, Khedr, and Shoukry 2019; Tran et al. 2019; Huang et al. 2019; Dutta, Chen, and Sankaranarayanan 2019) or hybrid systems (Ivanov et al. 2021). Here we follow up on recent work in the AI planning community (Vinzent, Steinmetz, and Hoffmann 2022; Vinzent, Sharma, and Hoffmann 2023), which tackles neural policies taking discrete action choices in non-deterministic state spaces, via **policy predicate abstraction (PPA)**. PPA builds an over-approximating abstraction not of the full state space Θ, but of the state-space subgraph containing only those transitions taken by π. While building that abstraction, for each possible abstract state transition $(s_{\mathcal{P}}, l, s'_{\mathcal{P}})$, an SMT solver is queried to decide whether π selects the necessary label l for at least one corresponding concrete state transition (s, l, s'). This SMT query is dispatched to *Marabou* (Katz et al. 2019), an SMT solver tailored to neural network analysis.

Here we extend this approach to another family of learned action policies, namely ones represented by tree ensembles (Breiman 2001; Friedman 2001; Chen and Guestrin 2016) which have proven to be powerful predictors in many ML problems (Grinsztajn, Oyallon, and Varoquaux 2022). Our key observation is that PPA is in principle agnostic to the policy representation, provided there exists a reasoning mechanism to which the abstract state transition queries can be dispatched. Given this, we can leverage progress on reasoning about other kinds of ML models. Here, we show how to do this for the **Veritas** tool which excels in analyzing tree-ensemble predictions (Devos, Meert, and Davis 2021; Cascioli, Devos, and Davis 2023). *We thus establish the first machinery for safety verification of tree ensemble policies.*

A key challenge in using Veritas within PPA is coping with the fact that tree ensembles are typically applied to static ML prediction problems, for which Veritas was designed. Capturing the dynamics of planning requires encoding constraints that capture how the abstract state changes when an action is applied. Yet the query language supported by Veritas (in its state-of-the-art implementation for multi-class classifiers) is limited to box constraints. We show how to encode the richer constraints necessary for PPA into additional trees added to the policy ensemble.

We evaluate our approach on Vinzent et al.'s (2022; 2023) benchmarks. Specifically, we use neural policies as teachers for imitation learning of tree-ensemble policies. We find that (1) tree ensembles can learn highly competitive policies, with the best tree ensembles achieving similar average reward as the neural policies, and being safe whenever the neural policy is. At the same time, (2) these tree-ensemble policies can be orders of magnitude faster to verify.

2 Background

Policy Safety Verification and PPA We consider transition systems described by a tuple $\langle \mathcal{V}, \mathcal{L}, \mathcal{O} \rangle$. $\mathcal{V}$ is a finite set of state variables. For each $v \in \mathcal{V}$ the domain D_v is a bounded integer interval. $\mathcal{L}$ is a finite set of **action labels**. $\mathcal{O}$ is a finite set of **operators**. Φ denotes the set of **linear constraints** over $\mathcal{V}$, i.e., of the form $d_1 \cdot v_1 + \cdots + d_r \cdot v_r \bowtie c$ with coefficients $d_1, \ldots, d_r, c \in \mathbb{Z}$, $\bowtie \in \{\leq, =, \geq\}$, and Boolean combinations thereof. An **operator** $o \in \mathcal{O}$ is a tuple (g, l, u) with **label** $l \in \mathcal{L}$, **guard** $g \in \Phi$ and **update** u over $\mathcal{V}$, where $u(v)$ is a linear assignment $d_1 \cdot v_1 + \cdots + d_r \cdot v_r + c$.

The **state space** of $\langle \mathcal{V}, \mathcal{L}, \mathcal{O} \rangle$ is a transition system $\Theta = \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$. The set of **states** $\mathcal{S}$ is the set of complete variable assignments over $\mathcal{V}$. The set of **transitions** $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$ contains (s, l, s') iff there exists an operator $o = (g, l, u)$ such that $g(s)$ evaluates to true, also written $s \models g$, and, for each $v \in \mathcal{V}$, $s'(v) = u(v)(s)$, also abbreviated $s' \models u(s)$.

An **action policy** π is a function $\mathcal{S} \rightarrow \mathcal{L}$. A **safety property** is a pair (ϕ_0, ϕ_u), where $\phi_0 \in \Phi$ identifies the set of start states and $\phi_u \in \Phi$ identifies the set of unsafe states. A policy π is **unsafe** with respect to (ϕ_0, ϕ_u) iff there exist states $s^0, s^u \in \mathcal{S}$ such that $s^0 \models \phi_0$, $s^u \models \phi_u$, and s^u is reachable from s^0 under π. Otherwise π is **safe**.

Policy predicate abstraction (PPA) (Vinzent, Steinmetz, and Hoffmann 2022) is an abstraction technique that, unlike classical predicate abstraction (Graf and Saïdi 1997), abstracts not the full state space Θ, but the subgraph induced by π. Given a set of **predicates** $\mathcal{P} \subseteq \Phi$, an **abstract state** $s_{\mathcal{P}}$ is a (complete) truth value assignment over $\mathcal{P}$. $[s_{\mathcal{P}}] = \{s \in \mathcal{S} \mid \forall p \in \mathcal{P}\colon p(s) = s_{\mathcal{P}}(p)\}$ denotes the set of all concrete states represented by $s_{\mathcal{P}}$. The **policy predicate abstraction** of Θ over $\mathcal{P}$ and π is the labeled transition system $\Theta^{\pi}_{\mathcal{P}} = \langle \mathcal{S}_{\mathcal{P}}, \mathcal{L}, \mathcal{T}^{\pi}_{\mathcal{P}} \rangle$ where $\mathcal{S}_{\mathcal{P}}$ is the set of all abstract states over $\mathcal{P}$ and $(s_{\mathcal{P}}, l, s'_{\mathcal{P}}) \in \mathcal{T}^{\pi}_{\mathcal{P}}$ iff there exists $(s, l, s') \in \mathcal{T}$ such that $s \in [s_{\mathcal{P}}]$, $s' \in [s'_{\mathcal{P}}]$ and $\pi(s) = l$.

If π is safe in $\Theta^{\pi}_{\mathcal{P}}$, then it is safe in Θ. An unsafe path in $\Theta^{\pi}_{\mathcal{P}}$ may be **spurious**, i.e., there does not exist a corresponding path in Θ under π. PPA with **counterexample-guided abstraction refinement** (CEGAR) (Clarke et al. 2003; Vinzent, Sharma, and Hoffmann 2023) can be used to iteratively remove such spurious abstract paths by refining $\mathcal{P}$, until either $\Theta^{\pi}_{\mathcal{P}}$ is proven safe, or a non-spurious abstract unsafe path is found, proving unsafety for π.

To compute $\Theta^{\pi}_{\mathcal{P}}$, one must solve the **abstract transition problem** for every possible abstract transition: $(s_{\mathcal{P}}, l, s'_{\mathcal{P}}) \in \mathcal{T}_{\mathcal{P}}$ iff for some operator (g, l, u) there exist concrete states $s \in [s_{\mathcal{P}}]$ and $s' \in [s'_{\mathcal{P}}]$ such that $s \models g$, $s' \models u(s)$ and $\pi(s) = l$. In classical predicate abstraction where no policy is considered and, hence, the condition $\pi(s) = l$ is dropped, such abstract transition problems are routinely encoded into satisfiability modulo theories (SMT) (Barrett and Tinelli 2018). For PPA however, the policy condition $\pi(s) = l$ introduces a key new source of complexity. Previous work (Vinzent, Steinmetz, and Hoffmann 2022) approaches this problem for action policies represented by neural networks. In this work we delve into PPA for safety verification of tree ensemble policies.

Additive Tree Ensembles A **binary decision tree** T over an input space $\mathcal{X} \subseteq \mathbb{R}^k$ is a tree whose inner nodes are associated with a branching condition $x_i < \alpha$, where x_i references one position in the input vector and $\alpha \in \mathbb{R}$ is a constant. Each leaf node is associated with a leaf value $\nu \in \mathbb{R}$. For an input $\vec{x} \in \mathcal{X}$, $T(\vec{x})$ is the value of the unique leaf reached by traversing the tree, following the left child of an inner node if the branching condition is satisfied by $\vec{x}$ and the right child otherwise. An **additive tree ensemble** (tree ensemble for short) is a sum of binary trees $\boldsymbol{T}(\vec{x}) := \sum_{i=1}^{n} T^i(\vec{x})$. We consider gradient boosted multi-class classifiers, which associate each class C with a tree ensemble $\boldsymbol{T}_C$ (Chen and Guestrin 2016); as well as multi-class random forests (Breiman 2001), where the leaf values in a single ensemble are vectors storing the probability of each class. Denoting the ensemble output for class C as $\boldsymbol{T}_C$, we define class predictions through $\mathcal{C}(\vec{x}) := argmax_C \boldsymbol{T}_C(\vec{x})$.

The Veritas Tool Verifiying a classifier $\mathcal{C}$ requires to determine the existence of an input $\vec{x}$ from a constrained region $\vec{x} \models \Gamma$ where Γ implements the constraints, s.t. $\mathcal{C}(\vec{x}) = C^t$ for the target class C^t. **Veritas**[1] is a state-of-the-art verification tool for tree-ensemble classifiers (Devos, Meert, and Davis 2021; Cascioli, Devos, and Davis 2023), which casts this decision problem as the optimization problem

$$\max_{\vec{x}} f^*(\vec{x}) \qquad \text{subject to} \quad \vec{x} \models \Gamma,$$
$$f^*(\vec{x}) := \left[\boldsymbol{T}_{C^t}(\vec{x}) - \max_{C \neq C^t} \boldsymbol{T}_C(\vec{x}) \right] \tag{1}$$

The decision problem is satisfied iff (1) has a solution with positive objective value $f^*(\vec{x}) > 0$. Veritas requires Γ to be a conjunction of **box constraints**, conditions $l_{x_i} \leq x_i \leq u_{x_i}$ where $l_{x_i}, u_{x_i} \in \mathbb{R} \cup \{\infty\}$. It solves (1) via heuristic search in the space of subsets of $\mathcal{X}$ representable by box constraints, starting from Γ and incrementally refining the constraint bounds until the desired concrete $\vec{x}$ is found.

3 Policy Predicate Abstraction for Tree Ensembles

We consider action policies π represented by $|\mathcal{L}|$-class tree-ensemble classifiers taking the state variables as input. For gradient-boosted tree ensembles, in what follows the tree ensemble associated with operator label l is denoted $\boldsymbol{T}_l$. For random forests, we denote by $\boldsymbol{T}_l$ the (single) tree ensemble with the leaf vectors projected onto dimension l.

To verify the safety of such π, we observe that PPA in principle applies to arbitrary policy representations, provided a method solving the abstract policy-transition problem is available. Here we show how to realize this potential for tree-ensemble policies.

Let $\mathcal{P} \subseteq \Phi$ be a set of predicates, and let $(s_{\mathcal{P}}, l, s'_{\mathcal{P}})$ be an abstract transition candidate. The conditions under which $(s_{\mathcal{P}}, l, s'_{\mathcal{P}}) \in \mathcal{T}^{\pi}_{\mathcal{P}}$ can in principle be encoded into SMT. However, using generic SMT solvers severely limits scalability. Instead one needs reasoning methods specialized to the type of ML model. For neural networks, Vinzent et al. (2022) leverage the Marabou framework (Katz et al. 2019). For tree ensembles, here we show how to leverage Veritas. The key challenge in doing so is encoding the necessary constraints into the input language supported by Veritas.

Encoding Linear Constraints as Additional Trees The basic component underlying all conditions in the abstract transition test are linear constraints which are strictly more general than the box constraints supported by Veritas. We

[1] https://github.com/laudv/veritas

first consider handling a single linear constraint, before proceeding to the whole transition test below. Let Γ be a conjunction of box constraints, let $\phi := d_1 v_1 + \dots d_r v_r \bowtie c$ be a linear constraint, and let $l \in \mathcal{L}$ be any target operator label. The key observation is that ϕ can be converted into an additional tree ensemble $\boldsymbol{T}_\phi$ such that $\exists s \models \Gamma \wedge \phi$ with $\pi(s) = l$ iff $\exists s' \models \Gamma$ with $f^*_{\pi'}(s') > 0$ in Veritas' objective (1) for the modified tree-ensemble classifier where $\boldsymbol{T}_l$ is replaced by the union with $\boldsymbol{T}_\phi$ (yielding the sum $\boldsymbol{T}_l + \boldsymbol{T}_\phi$).

Let $\nu^T_{\max}$ denote the maximal leaf value in a binary tree T, $\nu^T_{\min}$ denote the minimal leaf value, and let $\nu^{\boldsymbol{T}}_{\max} := \max_{l' \in l} \sum_{T \in \boldsymbol{T}_{l'}} \nu^T_{\max}$ and similarly $\nu^{\boldsymbol{T}}_{\min}$ be upper and lower bounds on the maximal and minimal possible value of all the tree ensembles. To generate the desired tree ensemble $\boldsymbol{T}_\phi$, we first enumerate all assignments to $v_1, \dots, v_r$ violating $d_1 v_1 + \dots d_r v_r \bowtie c$. This is exponential in the number of variables r in the constraint, which limits scalability in general, but works for small r as is typically the case in the PPA setting. For each violating assignment α, we then add to $\boldsymbol{T}_\phi$ a binary tree T^α that evaluates to $T^\alpha(s) = \nu^{\boldsymbol{T}}_{\min} - \nu^{\boldsymbol{T}}_{\max} =: \delta$ if s agrees with α, and $T^\alpha(s) = 0$ otherwise.

Given that every s can agree with at most one violating assignment α, we have $\boldsymbol{T}_\phi(s) \in \{0, \delta\}$, and $\boldsymbol{T}_\phi(s) = 0$ iff $s \models \phi$ holds by construction. This means in particular that $f^*_{\pi'}(s) = f^*_\pi(s)$ for all s s.t. $s \models \phi$. In other words, for every s with $s \models \Gamma \wedge \phi$ and $\pi(s) = l$, it holds that $f^*_{\pi'}(s) > 0$. Vice versa, for $s \not\models \phi$, the choice of δ ensures that

$$\begin{aligned} f^*_{\pi'}(s) &= \boldsymbol{T}_l(s) + \boldsymbol{T}_\phi(s) - \max_{l' \in \mathcal{L}, l \neq l'} \boldsymbol{T}_{l'}(s) \\ &= \boldsymbol{T}_l(s) - \nu^{\boldsymbol{T}}_{\max} + \nu^{\boldsymbol{T}}_{\min} - \max_{l' \in \mathcal{L}, l \neq l'} \boldsymbol{T}_{l'}(s) \\ &\leq \boldsymbol{T}_l(s) - \nu^{\boldsymbol{T}}_{\max} \\ &\leq 0 \end{aligned}$$

Combining both observations yields the desired result:

Theorem 1. *It holds for every conjunction of box constraints Γ, every linear constraint $\phi \in \Phi$, and tree-ensemble policy π that $\exists s \models \Gamma \wedge \phi$ with $\pi(s) = l$ iff $\exists s' \models \Gamma$ with $f^*_{\pi'}(s') > 0$ where π' replaces $\boldsymbol{T}_l$ with $\boldsymbol{T}_l + \boldsymbol{T}_\phi$.*

The compilation can be extended to conjunctions $\phi_1 \wedge \dots \wedge \phi_n$ of linear constraints, using the observation that $\sum_{i=1}^n \boldsymbol{T}_{\phi_i}(s) \leq \delta$ as soon as $s \not\models \phi_i$ for any $1 \leq i \leq n$; while being 0 if all constraints are satisfied. Hence, substituting the target tree ensemble $\boldsymbol{T}_l$ by the union with all the tree ensembles $\boldsymbol{T}_{\phi_1}, \dots, \boldsymbol{T}_{\phi_n}$ achieves what we want:

Corollary 2. *It holds for every conjunction of box constraints Γ, linear constraints $\phi_1, \dots, \phi_n \in \Phi$, and tree-ensemble policy π that $\exists s \models \Gamma \wedge \bigwedge_{i=1}^n \phi_i$ with $\pi(s) = l$ iff $\exists s' \models \Gamma$ with $f^*_{\pi'}(s') > 0$ where π' replaces $\boldsymbol{T}_l$ with $\boldsymbol{T}_l + \sum_{i=1}^n \boldsymbol{T}_{\phi_i}$.*

Encoding the Abstract Policy-Transition Test The abstract policy transition test needs to determine whether $(s_\mathcal{P}, l, s'_\mathcal{P}) \in \mathcal{T}^\pi_\mathcal{P}$. In addition to the encoding of linear constraints, this requires dealing with the dynamics of planning, involving two distinct abstract states (start and target) instead of a single input region as in the static settings for which Veritas was developed. We formulate the constraints on the target state over a *primed* copy of the state variables, as is commonly done in many settings. We handle these primed variables in Veritas queries by (temporarily) adding them as additional input variables to the tree ensembles.

Specifically, to determine whether $(s_\mathcal{P}, l, s'_\mathcal{P}) \in \mathcal{T}^\pi_\mathcal{P}$, we process each l-labeled operator (g, l, u) in turn, making an individual Veritas query for each. The box constraints Γ represent the state variable bounds and are the same in all queries. For each operator (g, l, u), we augment l's tree ensemble $\boldsymbol{T}_l$ via the tree-ensemble compilation of the conjunction of linear constraints given by $s_\mathcal{P}$, $s'_\mathcal{P}$, as well as the guard and update of (g, l, u), using the original or primed state variables within these constraints as appropriate. Within each such Veritas query, the primed variables are handled as additional inputs to the tree ensembles.

Thanks to Corollary 2 we have that, if, for the augmented $\pi_{(g,l,u)}$, Veritas finds a solution s, s' satisfying $f^*_{\pi_{(g,l,u)}}(s, s') > 0$, then $(s_\mathcal{P}, l, s'_\mathcal{P}) \in \mathcal{T}^\pi_\mathcal{P}$ as desired. If, on the other hand, no s, s' exist such that $f^*_{\pi_{(g,l,u)}}(s, s') > 0$ for any l-labeled operator (g, l, u), then $(s_\mathcal{P}, l, s'_\mathcal{P}) \notin \mathcal{T}^\pi_\mathcal{P}$:

Theorem 3. *$(s_\mathcal{P}, l, s'_\mathcal{P}) \in \mathcal{T}^\pi_\mathcal{P}$ iff there exists an operator (g, l, u), and s and s' with $s \models \Gamma$ and $s' \models \Gamma$ such that $f^*_{\pi_{(g,l,u)}}(s, s') > 0$ where $\pi_{(g,l,u)}$ replaces $\boldsymbol{T}_l$ by the union with the tree-ensemble compilation of the conjunction of $s_\mathcal{P}$, $s'_\mathcal{P}$, g, and u.*

4 Experiments

Our implementation is based on Vinzent et al.'s (2022; 2023) C++ code base realizing PPA and CEGAR for neural policies.[1] All experiments were run on Intel Xeon E5-2650v3 CPUs using time and memory cutoffs of 12h and 4GB.

Benchmarks A benchmark for policy verification is a pair of a transition system (modeled in the Jani language (Budde et al. 2017)) and a policy. The tree-ensemble policy training is described below. In terms of Jani models, we consider Vinzent et al.'s (2022; 2023) modified non-deterministic versions of Blocksworld, 8-puzzle, and Transport. Blocksworld and 8-puzzle come in two versions, "cost-aware (CA)" and "cost-ignore (CI)" that distinguish whether or not the policy receives as input a part of the state variables determining action cost. For Transport, we furthermore created a domain version in which we implemented a feature "number of packages to the left of the truck position" as an additional state variable. This was needed to obtain reasonably-performing tree-ensemble policies, highlighting the advantage of neural networks in automatic feature extraction.

Tree-Ensemble Policy Learning We learn tree ensemble policies through imitation learning, using for each benchmark a pretrained neural policy as the teacher (selection is described below). The training set is generated by simulating the teacher policy for 5000 start states, collecting all state-action pairs on these runs. Keeping the training set fixed, we trained multiple tree-ensemble policies with different hyperparameters. We used XGBoost (Chen and Guestrin 2016) (briefly: GB) as well as random forests (Breiman 2001)

[1]Source code and experiment data will be made public.

(briefly: RF). We varied tree depth across $4, 6, 8, 10, 15$; the number of trees across $5, 10, 20, 30$; and the learning rate across $0.4, 0.6, 0.8$. To choose among the trained policies, we evaluated them through simulation runs from 10000 randomly chosen start states (the testing set; same for all policies), considering three metrics: **reward**, the reward function used for neural policy training; **GoalFrac**, the fraction of simulation runs where the policy reached a goal state; and **fidelity**, the fraction of states for which the tree-ensemble policy selects the same action as its teacher.

For each of GB and RF, we select the tree ensemble that achieves the highest reward, discarding policies that were already found to be unsafe during the simulation runs. If several policies obtain the highest reward, we break ties by GoalFrac, depth, and then number of trees, thus encouraging selecting smaller tree ensembles.

Teacher Policy Selection. We considered the neural policies used by Vinzent et al. (2022; 2023) in their experiments, as well as newly trained ones. Vinzent et al. trained 3 policies for each Jani model, with hidden layer size 16, 32, and 64. Following the rationale of obtaining high-quality tree-ensemble policies, we chose as the teachers only the best-performing ones. Specifically, we measured the neural policies' quality through simulation runs using the same testing set as for the tree-ensemble policies, and selected for training all those whose average reward is within 1% of that of the best performing policy. We discarded neural policies found to be unsafe during these runs.[2]

Beyond the neural policies from previous work, we created some additional policies. Namely, as Vinzent et al.'s neural policies for Transport and 8-puzzle are of poor quality (never reaching the goal), we also invested effort in finding better neural policies, through supervised learning from domain-specific solvers we constructed.

Results Table 1 summarizes our results. Regarding policy quality, fidelity is 1 or almost 1 in most cases, and performance in terms of GoalFrac and average reward is at least as good as that of the NN teacher in most cases. The most notable exception is Transport, where the tree ensembles need an additional feature to perform well.

The tree-ensemble policies also preserve safety. To the extent we can verify it, the best tree-ensemble policies are safe whenever the neural teacher policy is. Random forests are unsafe in 8Blocks CI, but are the only safe policies in Transport+Feature. Regarding verification efficacy, in Blocksworld the results are very positive, with tree-ensemble policy verification outperforming neural policy verification by up to 3 orders of magnitude. In 8-puzzle, none of the policies can be proved safe or unsafe within the time/memory limits. In Transport, runtime varies a lot for the unsafe policies as, in finding an unsafe path, one may "get lucky". Most remarkably, RF policies can be proved safe.

[2]We remark that, on very bad policies that hardly ever reach the goal, tree-ensemble verification sometimes takes much longer than neural policy verification. It remains future work to determine why that is the case. Here, we focus on performant policies, for which investing the effort of verification actually makes sense.

5 Conclusion

We establish the first methodology for verification of tree-ensemble policies. Our results show that such verification can be feasible, and can be more effective than neural policy verification. Our results also show that tree-ensemble policies can be competitive in terms of performance, though they may suffer from lack of features as exemplified by the Transport benchmark. An important topic for future work is to address the latter through automatic feature generation (e.g. (Drexler and Seipp 2023)), and to extend PPA verification to deal with such features. Our hypothesis is that, *within the realm of problems where a full verification can be hoped for*, tree ensembles constitute a serious alternative to neural networks, and may thus, within that realm, offer a better trade-off between performance vs. verifiability. Much more work is needed to explore this hypothesis in full.

References

Barrett, C. W.; and Tinelli, C. 2018. Satisfiability Modulo Theories. In *Handbook of Model Checking*, 305–343. Springer.

Breiman, L. 2001. Random forests. *Machine learning*, 45: 5–32.

Budde, C. E.; Dehnert, C.; Hahn, E. M.; Hartmanns, A.; Junges, S.; and Turrini, A. 2017. JANI: Quantitative Model and Tool Interaction. In *TACAS*.

Cascioli, L.; Devos, L.; and Davis, J. 2023. Multi-class Robustness Verification for Tree Ensembles. In *Proceedings of the Verifying Learning AI Systems Workshop*.

Chen, T.; and Guestrin, C. 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794.

Clarke, E.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2003. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the Association for Computing Machinery*, 50(5): 752–794.

Devos, L.; Meert, W.; and Davis, J. 2021. Versatile Verification of Tree Ensembles. In *International Conference on Machine Learning*, 2654–2664. PMLR.

Drexler, D.; and Seipp, J. 2023. DLPlan: Description Logics State Features for Planning. In *ICAPS 2023 System Demonstrations and Exhibits*.

Dutta, S.; Chen, X.; and Sankaranarayanan, S. 2019. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *Proceedings of the 22nd International Conference on Hybrid Systems: Computation and Control (HSCC'19)*, 157–168.

Friedman, J. H. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189–1232.

Garg, S.; Bajpai, A.; et al. 2019. Size independent neural transfer for rddl planning. In *ICAPS*.

Graf, S.; and Saïdi, H. 1997. Construction of Abstract State Graphs with PVS. In *CAV*, 72–83.

Grinsztajn, L.; Oyallon, E.; and Varoquaux, G. 2022. Why do tree-based models still outperform deep learning on typical tabular data? In *Advances in Neural Information Processing Systems*, volume 35, 507–520.

Groshev, E.; Goldstein, M.; Tamar, A.; Srivastava, S.; and Abbeel, P. 2018. Learning generalized reactive policies using deep neural networks. In *ICAPS*.

Huang, S.; andf Wenchao Li, J. F.; Chen, X.; and Zhu, Q. 2019. ReachNN: Reachability analysis of neural-network controlled systems. *ACM Transactions on Embedded Computing Systems*, 18: 1–22.

Issakkimuthu, M.; Fern, A.; and Tadepalli, P. 2018. Training Deep Reactive Policies for Probabilistic Planning Problems. In *ICAPS*, 422–430.

Ivanov, R.; Carpenter, T. J.; Weimer, J.; Alur, R.; Pappas, G. J.; and Lee, I. 2021. Verifying the Safety of Autonomous Systems with Neural Network Controllers. *ACM Transactions on Embedded Computing Systems*, 20(1): 7:1–7:26.

Karia, R.; and Srivastava, S. 2021. Learning Generalized Relational Heuristic Networks for Model-Agnostic Planning. In *AAAI*, 8064–8073.

Katz, G.; Huang, D. A.; Ibeling, D.; Julian, K.; Lazarus, C.; Lim, R.; Shah, P.; Thakoor, S.; Wu, H.; Zeljic, A.; Dill, D. L.; Kochenderfer, M.; and Barrett, C. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In Dillig, I.; and Tasiran, S., eds., *Computer Aided Verification. CAV 2019*, LNCS 11561. Cham: Springer. https://doi.org/10.1007/978-3-030-25540-4_26.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M. A.; Fidjeland, A.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529: 484–503.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T.; Simonyan, K.; and Hassabis, D. 2018. A general reinforcement learning algorithm that masters Chess, Shogi, and Go through self-play. *Science*, 362(6419): 1140–1144.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022a. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In *ICAPS*.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022b. Learning Generalized Policies without Supervision Using GNNs. In *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning (KR'22)*.

Sun, X.; Khedr, H.; and Shoukry, Y. 2019. Formal Verification of Neural Network Controlled Autonomous Systems. In *International Conference on Hybrid Systems: Computation and Control (HSCC'19)*.

Toyer, S.; Thiébaux, S.; Trevizan, F. W.; and Xie, L. 2020. ASNets: Deep Learning for Generalised Planning. *Journal of Artificial Intelligence Research*, 68: 1–68.

Tran, H.; Cai, F.; Lopez, D. M.; Musau, P.; Johnson, T. T.; and Koutsoukos, X. D. 2019. Safety Verification of Cyber-Physical Systems with Reinforcement Learning Control. *ACM Transactions on Embedded Computing Systems*, 18(5s): 105:1–105:22.

Vinzent, M.; Sharma, S.; and Hoffmann, J. 2023. Neural Policy Safety Verification via Predicate Abstraction: CEGAR. In *AAAI*.

Vinzent, M.; Steinmetz, M.; and Hoffmann, J. 2022. Neural Network Action Policy Verification via Predicate Abstraction. In *ICAPS*.

Jani model	Policy	Fid	GoalFrac	Reward	Safe	VerTime
4Blocks CI	NN (16h)	—	1.00	172.5	Yes	4.8
	GB	1.00	1.00	172.5	Yes	2.0
	RF	1.00	1.00	172.5	Yes	2.3
	NN (32h)	—	1.00	172.0	Yes	13.6
	GB	1.00	1.00	172.0	Yes	1.7
	RF	1.00	1.00	172.0	Yes	2.4
	NN (64h)	—	1.00	172.9	Yes	81.1
	GB	1.00	1.00	172.9	Yes	2.2
	RF	1.00	1.00	172.9	Yes	2.0
6Blocks CI	NN (64h)	—	0.98	156.2	Yes	6745.0
	GB	0.99	0.98	156.8	Yes	95.6
	RF	1.00	0.97	155.6	Yes	167.6
8Blocks CI	NN (64h)	—	0.60	77.7	—	—
	GB	0.96	0.70	94.4	Yes	1750.5
	RF	0.90	0.85	121.2	No	1.5
4Blocks CA	NN (32h)	—	1.00	172.5	Yes	580.3
	GB	0.99	1.00	172.5	Yes	16.9
	RF	1.00	1.00	172.5	Yes	9.5
	NN (64h)	—	1.00	172.9	—	—
	GB	0.99	1.00	172.9	Yes	13.1
	RF	1.00	1.00	172.9	Yes	13.1
6Blocks CA	NN (64h)	—	0.95	150.2	—	—
	GB	0.99	0.96	152.0	Yes	293.3
	RF	1.00	0.97	153	—	—
8Blocks CA	NN (64h)	—	0.00	-22.6	—	—
	GB	0.84	0.00	-22.3	—	—
	RF	0.74	0	-22.6	—	—
8Puzzle CI	NN (256h)	—	0.09	14.1	—	—
	GB	0.93	0.07	10.0	—	—
	RF	0.99	0.08	13.3	—	—
Transport	NN (16h)	—	0.97	99.1	No	42.5
	GB	0.98	0.01	-2.0	No	500.0
	RF	1.00	0.79	65.6	—	—
Transport + Feature	NN (16h)	—	0.99	110.2	No	9450.2
	GB	1.00	0.99	110.3	No	853.0
	RF	1.00	1.00	93.4	Yes	2714.4
	NN (32h)	—	0.99	110.5	No	0.5
	GB	1.00	0.99	110.3	No	875.0
	RF	1.00	1.00	93.4	Yes	2729.9
	NN (64h)	—	0.99	110.2	—	—
	GB	0.99	0.99	109.4	—	—
	RF	1.00	0.99	93.3	Yes	0.6

Table 1: Results summary. Xh: hidden layer size X, GB: XGBoost, RF: random forests. Fid: fidelity, Safe: verification outcome, VerTime: total verification runtime in seconds.