Formula Neural Networks as Action Policies in Safety-Critical Planning: An Empirical Evaluation

Victor Scherer Putrich[1], **Tim Nico Bauerschmidt**[1], **Chaahat Jain**[1], **Songtuan Lin**[1], **Jörg Hoffmann**[1,2], **Isabel Valera**[1]

[1]Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
[2]German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany
{putrich, bauerschmidt, jain, songtuan.lin, hoffmann, valera}@cs.uni-saarland.de

Abstract

Formula neural networks (FNNs) are a recently proposed alternative to neural networks (NNs), with Boolean neuron values and hidden-layer nodes defined as propositional formulas over the previous layer. FNNs have been successfully applied to classification and, compared to standard NNs, offer greater interpretability. Here we show that they are also a promising alternative for learning action policies and that they can be more amenable to formal verification. We consider planning under uncertainty, with the objective to reach the goal while avoiding unsafe states. We adapt a recently proposed NN policy safety verification algorithm to FNNs. Our experiments show that, compared to NNs, FNNs can often learn competitive policies while being much faster to verify.

Introduction

Neural networks (NNs) are widely used to represent policies learned through reinforcement learning (RL), and have been highly successful in planning under uncertainty (Mnih et al. 2015; Silver et al. 2016). However, when such policies are used in safety-critical settings, their empirical performance must be complemented by formal safety guarantees. Existing verification methods can provide such guarantees for NN policies, but often at high computational cost.

In this paper, we explore an alternative representation for action policies. The models we consider follow the structure of feed-forward NNs, but their hidden-layer neurons represent logical formulas over binary inputs (Wang et al. 2021; Petersen et al. 2022; Xu, Walter, and Vreeken 2025). As no unified name exists for this family of models, we refer to them as formula neural networks (FNNs). They are trained through differentiable relaxations of logical operations and yield rule-based representations after discretization.

We place our work within a framework recently used in the investigation of policy predicate abstraction, a policy safety verification algorithm (Vinzent, Sharma, and Hoffmann 2023; Jain et al. 2024). Planning tasks here have initial state as well as action-outcome uncertainty, and use bounded-integer state variables. They are modelled as automata in the JANI language (Budde et al. 2017). For our setting, a policy must reach the goal while staying safe, i.e., avoiding states that satisfy a failure formula. After training of a policy, policy predicate abstraction is used as a posthoc verification process to check whether the policy is safe.

We experiment with different design variants of FNNs. For binarizing the bounded-integer states (which is done at the first FNN layer), we explore two variants: (1) a one-hot encoding of state-variable values, and (2) the conditions appearing in the JANI model. We train the FNNs by imitation learning from NN teachers learned by prior works, using either a classification or a regression loss. We adapt PPA to FNNs in a straightforward manner, by replacing an NN-specialized Satisfiability Modulo Theories (SMT) solver with a generic SMT solver.

Our findings are as follows. First, classification-trained FNNs typically match their neural teachers, sometimes achieving higher goal-reaching or lower failure rates. Regression-trained models also achieve this, but less robustly, with substantially worse performance than their teachers in some domains. On the other hand, regression-trained FNNs are verified safe more often than classification-trained ones. Second, PPA is much faster for FNNs than for their neural teachers, often by an order of magnitude or more. Third, using conditions from the JANI model instead of one-hot encoding yields considerably smaller FNNs that are more amenable to verification.

Note that, per the first observation, classification training vs. regression training have complementary virtues and the configuration yielding the best trade-off between FNN performance vs. safety depends on the domain. That is acceptable in our context however, as policies generalize over initial states and action outcomes and can thus be selected once for long-term use.

Background

We consider planning tasks with probabilistic action outcomes. A task is described by a finite set of state variables $\mathcal{V}$ and a finite set of action labels $\mathcal{A}$. For each variable $v \in \mathcal{V}$, the domain D_v is a non-empty bounded set of integers. A state is a complete assignment over $\mathcal{V}$, and we write $\mathcal{S}$ for the set of all such assignments (states). Each action $a \in \mathcal{A}$ has *guards* specifying when the action is applicable. Guards are boolean combinations of linear constraints over $\mathcal{V}$ of the form $\sum_{i=1}^{r} d_i v_i + c \bowtie 0$ for some $r \in \mathbb{N}$, $c \in \mathbb{Z}$, coefficients $d_i \in \mathbb{Z}$, variables $v_i \in \mathcal{V}$, and $\bowtie \in \{\leq, =, \geq\}$.

The semantics of the task is a Markov decision process

$\Theta = \langle \mathcal{S}, \mathcal{A}, \mathbb{P} \rangle$, where $\mathbb{P}(s, a, s')$ denotes the probability of reaching state s' after applying action a in state s. An action a is applicable in s iff its guard is satisfied in s; equivalently, iff there exists some $s' \in \mathcal{S}$ such that $\mathbb{P}(s, a, s') > 0$.

A *reach-avoid property* is a tuple $\langle \phi_0, \phi_G, \phi_F \rangle$, where ϕ_0, ϕ_G, and ϕ_F are linear integer constraints over $\mathcal{V}$ identifying the sets of initial, goal, and failure states, respectively. An *action policy* is a function $\pi : \mathcal{S} \to \mathcal{A}$. A policy path is a finite or infinite sequence $\sigma = s_0, a_0, s_1, a_1, \ldots$ such that $a_i = \pi(s_i)$ and $\mathbb{P}(s_i, a_i, s_{i+1}) > 0$ for all i. A policy is *unsafe* if there exists a policy path starting in an initial state $s_0 \models \phi_0$ that reaches a failure state before reaching a goal state, i.e., there exists some $n \geq 0$ such that $s_n \models \phi_F$ and $s_i \not\models \phi_G$ for all $0 \leq i < n$. Otherwise, the policy is *safe*. Note that *safety does not imply goal reaching*: a safe policy may also avoid both ϕ_G and ϕ_F by cycling.

Formula Neural Networks. We model action policies using the FNN architecture proposed by Wang et al. (2021, 2024); hereafter, FNN refers to this model, not to the general concept. As illustrated in Figure 1, FNNs follow the overall structure of standard neural networks, but hidden neurons represent propositional formulas rather than continuous activations. An FNN consists of a binarization layer, $L > 0$ logical layers, and a final linear output layer.

The binarization layer produces binary inputs $u^{(0)} \in \mathbb{B}^{n_0}$, where $\mathbb{B} = \{0, 1\}$ denotes the Boolean domain and $n_0 \in \mathbb{N}$ is the input dimension. Each logical layer maps a Boolean vector from the previous layer to another Boolean vector. Each neuron is either an AND (conjunction) or an OR (disjunction) neuron. A neuron i in layer ℓ selects a subset of inputs from the previous layer via a weight vector $W_i^{(\ell)} \in [0, 1]^{n_{\ell-1}}$. Let $pred(i) = \{j \mid W_{i,j}^{(\ell)} \geq 0.5\}$ denote the predecessors of neuron i. The neuron then computes

$$u_i^{(\ell)} = \begin{cases} \bigwedge_{j \in pred(i)} u_j^{(\ell-1)} & \text{if } i \text{ is an AND neuron} \\ \bigvee_{j \in pred(i)} u_j^{(\ell-1)} & \text{if } i \text{ is an OR neuron} \end{cases}.$$

So each neuron represents a propositional subformula over the binarized inputs $u^{(0)}$. The neurons $u_i^{(L)}$ in the last (logical) layer define full formulas ϕ_i over $u^{(0)}$, which are combined by a linear layer to produce a real-valued output vector.

FNNs can be trained with gradient descent using surrogate gradients for the discrete logical operations. Specifically, Wang et al. (2024) introduce differentiable activation functions $\tilde{u}_i^{(\ell)}(\cdot, W_i^{(\ell)})$ that approximate the logical neurons $u_i^{(\ell)}$. During the forward pass, the Boolean activations $u_i^{(\ell)}$ are computed according to their logical definitions. However, since these activations are non-differentiable with respect to both $W_i^{(\ell)}$ and $u^{(\ell-1)}$, backpropagation uses the gradients of the differentiable surrogates $\tilde{u}_i^{(\ell)}$ instead. Thus, for a loss function $\mathcal{L}$, gradients in each layer are propagated as

$$\frac{\partial \mathcal{L}}{\partial u^{(\ell-1)}} = \frac{\partial \mathcal{L}}{\partial u^{(\ell)}} \frac{\partial \tilde{u}^{(\ell)}}{\partial u^{(\ell-1)}}, \qquad \frac{\partial \mathcal{L}}{\partial W_i^{(\ell)}} = \frac{\partial \mathcal{L}}{\partial u_i^{(\ell)}} \frac{\partial \tilde{u}_i^{(\ell)}}{\partial W_i^{(\ell)}}.$$

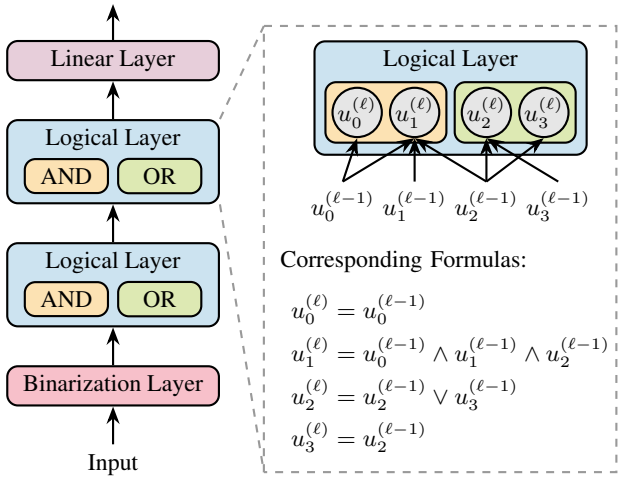

Figure 1: *FNN architecture with two hidden layers*, adapted from (Wang et al. 2021). Inputs are binarized and processed by logical hidden layers whose neurons learn propositional formulas over binary (Boolean) inputs $u^{(\ell-1)}$. In our setting, half of the neurons compute conjunctions and half compute disjunctions, followed by a standard linear output layer.

This enables learning the weights $W_i^{(\ell)}$, which determine the neurons from layer $\ell - 1$ that are included in the formula represented by $u_i^{(\ell)}$, using gradient-based optimization.

Safety Verification. Policy predicate abstraction has recently been proposed as an algorithm to verify whether a given learned policy is safe (Vinzent, Sharma, and Hoffmann 2023). Policy predicate abstraction builds an overapproximating abstraction defined through a set of predicates (linear constraints over state variables). Crucially, this overapproximation pertains to the state-space subgraph restricted to policy-induced transitions. Hence, if there does not exist a path from ϕ_0 to ϕ_F in the abstraction, then the policy is safe. Otherwise, the set of predicates is refined using CEGAR (Clarke et al. 2003) until the policy is proved safe or unsafe.

The key computational problem in policy predicate abstraction is deciding which transitions are valid in the abstraction; this is formulated as a Satisfiability Modulo Theories (SMT) query. Each query combines state-transition requirements defined by the planning task with an encoding of the policy function π to determine whether a policy-induced transition between states exists in the abstraction. Vinzent, Sharma, and Hoffmann (2023) use *Marabou* (Katz et al. 2019), an SMT solver specialized for NNs, to process these queries. Policy predicate abstraction is readily applicable to other learned policy representations, provided that a solver exists for the abstract transition queries.

FNN Policies

We construct FNN policies as follows. Given a state $s \in \mathcal{S}$, we define the binarization layer β that produces a Boolean feature vector $u^{(0)} := \beta(s) \in \mathbb{B}^{n_0}$. In our setting, β is in-

stantiated either by *one-hot features* or by JANI *features*. The former is the standard encoding of variable-value pairs, where each pair is represented by a distinct boolean feature. The latter is derived from the guards and reach-avoid constraints. More specifically, for each action guard and for the formulas defining ϕ_G and ϕ_F, we decompose conjunctions and disjunctions into subformulas and use the resulting set of constraints as features, such as $x = 3$, $y \leq 5$, or $x = y+1$.

The neurons in the last logical layer of the FNN can thus be interpreted as propositional formulas $\phi_1, \ldots, \phi_{n_L}$ over the state, whose truth values correspond to the outputs of that layer $\phi_i(s) \coloneqq u_i^{(L)}(s)$.

The linear layer then assigns each action $a \in \mathcal{A}$ a real-valued score given by a weighted sum of the formulas represented in the last logical layer:

$$score_a(s) = b_a + \sum_{i=1}^{n_L} W_{a,i}\, \phi_i(s),$$

where $W_{a,i} \in \mathbb{R}$ is the weight of formula $\phi_i \in \mathbb{B}$ for action a, and $b_a \in \mathbb{R}$ is a bias term. Learned policies may select any action in any state, even if it is not applicable. An established technique to avoid this is to filter the inapplicable actions prior to selecting an action (Toyer et al. 2020; Ståhlberg, Bonet, and Geffner 2022). Hence, the FNN policy decision is defined as the action with the highest score among the applicable actions. $app(s)$ in state s:

$$\pi_{fnn}(s) \coloneqq \arg\max_{a \in app(s)} score_a(s).$$

We train FNN policies with imitation learning using NN policies as teachers. For each problem, we sample traces from the teacher and collect states together with the teacher's output. We consider two types of FNN policies: *classifiers* trained to predict the teacher-selected action by minimizing the cross-entropy loss, and *regressors* trained to predict the full vector of action values returned by the teacher by minimizing the mean squared error. The training dataset is shuffled and split into five folds. In each run, one fold is used for testing, and 20% of the remaining data are held out for validation while the rest are used for training.

For post-hoc safety verification of FNN policies, we use policy predicate abstraction by Vinzent, Sharma, and Hoffmann (2023) as our verification method, starting from the authors' C++ codebase. To adapt verification to FNNs, we only need to replace the policy representation in policy predicate abstraction's abstract transition queries. To this end, we use a straightforward encoding of FNNs into SMT, and call Z3 (de Moura and Bjorner 2008) to solve the resulting queries.

Experiments

To train FNNs for our setting we adapted the codebase of Wang et al. (2024). The code for all experiments is available on our repository *https://github.com/vsp-ai/socs26-repo*. We ran experiments comparing (1) the policy quality and (2) the verifiability of FNNs to their neural teachers. In what follows, we describe our experiment setup and discuss the main results. Training was performed on NVIDIA A100 and P100 GPU nodes. Evaluation and verification experiments were conducted on machines with AMD EPYC 9654 processors at 2.4GHz with time and memory limits of 24 hours and 16GB respectively.

Benchmarks We use 26 benchmark instances from Jain et al. (2025), each defined by a JANI model and a trained NN policy on that model. These benchmarks have been modified by prior work, with initial-state and action-outcome non-determinism added. We focus on discrete domains with bounded integer-valued variables. The JANI models are probabilistic versions of the planning benchmarks *Blocksworld* and *Transport*; an abstract version of a logistics problem at Airbus *Beluga*; as well as variants of a transportation domain, *one way line (1WL)* and *two way line (2WL)*, modelling a truck moving along a line of locations with a chance of slipping on icy regions. The state spaces of these benchmark instances are huge ($> 10^8$ except on a beluga instances); Figure 2 shows the data. Therefore, naive algorithms to find or verify policies are infeasible.

The neural policies are feedforward NNs, trained with Q-learning to maximize the expected reward of reaching the goal. To obtain good teachers on benchmarks where the neural policy performs poorly, we trained new NN policies using Q-learning and selected the ones with the highest average reward on sample runs from $10,000$ sampled start states.

Experiment Setup To train the FNNs, we fix the number of logical layers to 1 and vary hyperparameter settings for the binarization layer (one-hot and JANI features), the learning rate $\{0.01, 0.005\}$ and the width of the logical layer $\{128, 256, 512\}$. To choose among the trained policies, we evaluate them through simulation runs from a fixed set of $50,000$ sampled start states. We consider two metrics: (1) **%Goal** the percentage of runs in which the policy successfully reaches a goal state and (2) **%Fail** the percentage of runs where the policy reaches a failure state. Note that neither outcome is guaranteed as a policy may cycle indefinitely without reaching either.

Policy Selection In the targeted application scenario, it is quite practical for the user to make hyperarameter choices *on a per-instance basis*. After all, the policy is designed to be used for many runs on a given instance (exponentially many possible initial states and action-outcome combinations). Thus, there is no need to identify a single hyperparameter setting that works uniformly across all domains. In the following results analyses, we therefore select policies per-instance, choosing suitable hyperparameter settings according to sensible user preferences. We state the used preferences along with each result presentation.

Results

Here we discuss the main results of our experiments. We focus on three questions: (1) Can FNNs match the performance of their neural teachers? (2) Are FNNs easier to verify compared to their neural teachers? (3) Does the binarization layer make a difference?

Q1. Can FNNs match the performance of their neural teachers? Figure 2 shows the FNN performance relative

Figure 2: *FNN policy quality relative to NN teachers.* Candidate FNNs use one logical layer with $\{128, 256, 512\}$ logical nodes, binarization layer using $\{$ one-hot, JANI $\}$ features, and learning rate in $\{0.01, 0.005\}$. For each benchmark and training objective, we select the FNN maximizing %Goal, then minimizing %Fail, with ties broken by smaller width. Labels ($\geq |S|$, $\%Goal_{\mathrm{NN}}$, $\%Fail_{\mathrm{NN}}$) show state-space lower bounds and teacher performance; for teacher performance, $> 0\%$ means positive but below 1%. Student performance is shown as deviations from the teacher: ΔGoal is $\%Goal_{\mathrm{FNN}} - \%Goal_{\mathrm{NN}}$, and ΔFail is $\%Fail_{\mathrm{FNN}} - \%Fail_{\mathrm{NN}}$, with deviations annotated. Positive ΔGoal and negative ΔFail are desirable.

to NN teachers on all 26 benchmarks. For classification-trained FNNs (left-hand side of Figure 2), the answer is yes. The best FNNs offer similar $\%Goal$ and $\%Fail$ as their neural teachers. They are rarely worse, and sometimes substantially better (e.g. on benchmark 3, `1WL Det (40, 25)`, where the FNN reaches the goal in about 20 percentage points more sample runs). For regression-trained FNNs (right-hand side), the results are more limited. On some benchmarks, goal-reaching performance is much lower and failure higher than the teacher.

Q2. Are FNNs easier to verify compared to their neural teachers? The answer is yes as shown in Figure 3. Verification of the neural teachers was feasible on only 5 of 26 benchmarks within the 24-hour time limit. In contrast, verification for classification-trained and regression-trained FNNs completed in 12 and 16 instances, respectively, often an order of magnitude faster than their teachers (points clearly below the line in the log-log plot).

Most of the FNN verification outcomes are unsafe, in particular for classification-trained FNNs. For regression-trained FNNs, there are 4 proved-safe cases, like for NNs. In particular, the regression-trained FNN for `Transport` is verified safe despite the neural teacher itself being unsafe. So regression training seems better at finding safe policies. The runtime advantage of our verification for FNN vs. NN policies is also large: the largest improvements are from 1523s to 35s and from 22528s to 43s.

One may wonder whether the neural teachers of the unsafe FNN policies are safe. This is not the case: in 18 of the 21 instances on which the teachers timed out, we found unsafe policy runs by sampling.[1]

Q3. Does the binarization layer make a difference? As shown in Table 1, the answer is yes: JANI features reduce average rule size by around a factor of 5. This directly benefits our verification approach because its computational cost is linked to FNN's size (i.e., the size of the learned propositional rules). Consequently, the smaller rules reduce timeouts by 5 for classification-trained and by 11 for regression-trained FNNs.

Table 1 also gives a results overview, distinguishing classification training vs. regression training, with data for policy quality and verification counts. The results, together with our observations above, reveal a clear trade-off between pol-

[1]While this means that, for many of the NN teachers, unsafety is actually easy to prove by other means, the consistent runtime advantage of PPA for FNN vs. NN policies is still meaningful. In particular, the most drastic runtime improvements pointed out above – from 1523s to 35s and from 22528s to 43s – are for (regression-trained) FNN policies proved safe. Furthermore, sampling is not a verification method, it can only show the presence of unsafe runs not their absence. The state spaces of these benchmarks are huge, cf. above; exhaustive enumeration is infeasible.

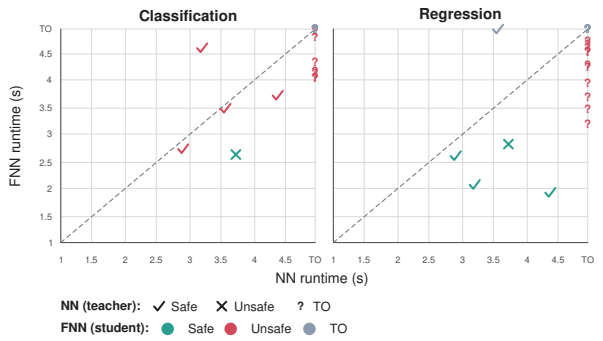

Figure 3: *Verification runtimes for FNN policies compared to their neural teachers.* For each benchmark and training objective, we select the best FNN model per instance by prioritizing verification outcomes: safe $>$ unsafe $>$ timeout (TO), followed by higher %Goal, lower %Fail, and smaller width. Candidate FNNs use one logical layer with either one-hot or JANI binarization, learning rate in $\{0.01, 0.005\}$, and logical-layer width in $\{128, 256, 512\}$. Axes show runtime in seconds on a $\log_{10}$ scale; TO is the 24-hour timeout. Shape marks the NN outcome, while color marks whether the selected FNN is safe, unsafe, or times out.

icy quality and verifiability. While classification preserves or improves policy quality, it is more difficult to verify. In contrast, regression is better at learning to preserve safety when the teacher policy is safe, as it gets a training signal on the magnitude of differences between actions, in particular negative values on unsafe actions. However, the capability of FNN to faithfully model the action scores of an NN is limited. In some domains, this leads to worse goal-reaching performance.

As argued above however, this lack of a single configuration that works well across all domains is not a major practical problem in our setting. As policies will be used long-term, hyperparameter selection can be done on a per-instance basis. For example, if the user selects the best FNN policy per instance, prioritizing safety, then maximizing %Goal and minimizing %Fail, the aggregate values as in Table 1 are %Goal 82, %Fail 5, 4 safe, 7 unsafe, and 15 timeouts.

Conclusion

Neural policies are widely used for planning problems, but their verification remains challenging. Our results show that FNNs can serve as action policies while being faster to verify. In particular, using JANI features in the binarization layer yields much smaller rules, improves verifiability, and can preserve or improve policy quality.

More broadly, our results open the path to deeper investigation of FNNs in planning and motivate the search for alternative policy representations whose structure facilitates verification.

			Evaluation		Verification		
Model	Obj.	$\lvert\phi\rvert$	%Goal	%Fail	✓	×	TO
NN	–	–	83	6	4	1	21
FNN (OH)	Class.	47	81	6	0	7	19
FNN (J)	Class.	6	82	5	1	11	14
FNN (OH)	Reg.	65	52	18	2	3	21
FNN (J)	Reg.	8	57	12	4	12	10

Table 1: *Effect of binarization, and results overview.* For each benchmark, we group FNNs by training objective and binarization layer. Labels OH and J denote one-hot and JANI features, respectively. Within each group, we select the best FNN model per instance by prioritizing verification outcomes: safe (✓) $>$ unsafe ($\times$) $>$ timeout (TO), followed by higher %Goal, lower %Fail, and smaller width. We report the training objective (Obj.), average literals per rule ($\lvert\phi\rvert$), policy quality (%Goal, %Fail), and verification counts.

Acknowledgments

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – GRK 2853/1 "Neuroexplicit Models of Language, Vision, and Action" - project number 471607914. This work was funded by DFG Grant 389792660 as part of TRR 248 (CPEC, https://perspicuous-computing.science).

References

Budde, C. E.; Dehnert, C.; Hahn, E. M.; Hartmanns, A.; Junges, S.; and Turrini, A. 2017. JANI: Quantitative Model and Tool Interaction. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'17)*, 151–168.

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.

de Moura, L.; and Bjorner, N. 2008. Z3: An Efficient SMT Solver. In Ramakrishnan, C.; and Rehof, J., eds., *Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2008*, LNCS 4963. Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-540-78800-3_24.

Jain, C.; Cascioli, L.; Devos, L.; Vinzent, M.; Steinmetz, M.; Davis, J.; and Hoffmann, J. 2024. Safety Verification of Tree-Ensemble Policies via Predicate Abstraction. In *Proceedings of the 27th European Conference on Artificial Intelligence (ECAI'24)*.

Jain, C.; Sherbakov, D.; Vinzent, M.; Steinmetz, M.; Davis, J.; and Hoffmann, J. 2025. Policy Safety Testing in Non-Deterministic Planning: Fuzzing, Test Oracles, Fault Analysis. In *Proceedings of the 28th European Conference on Artificial Intelligence (ECAI'25)*.

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.

Petersen, F.; Borgelt, C.; Kuehne, H.; and Deussen, O. 2022. Deep differentiable logic gate networks. *Advances in Neural Information Processing Systems*, 35: 2006–2018.

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.

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

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.

Vinzent, M.; Sharma, S.; and Hoffmann, J. 2023. Neural Policy Safety Verification via Predicate Abstraction: CEGAR. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI'23)*, 15188–15196. AAAI Press.

Wang, Z.; Zhang, W.; Liu, N.; and Wang, J. 2021. Scalable Rule-Based Representation Learning for Interpretable Classification. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y. N.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, 30479–30491.

Wang, Z.; Zhang, W.; Liu, N.; and Wang, J. 2024. Learning Interpretable Rules for Scalable Data Representation and Classification. *IEEE Trans. Pattern Anal. Mach. Intell.*, 46(2): 1121–1133.

Xu, S.; Walter, N. P.; and Vreeken, J. 2025. Neural Rule Lists: Learning Discretizations, Rules, and Order in One Go. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.