δ-APPROXIMATE STRUCTURAL TRANSFORMATION OF DECISION TREES INTO FIXED-DEPTH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural networks (NNs) are effective decision-making models, but their decision logic is often difficult to inspect and verify. A large body of research has focused on converting NNs into interpretable models to facilitate their analysis and modification. Decision trees are an attractive representation since they expose their input space partitions and can therefore be used to understand the decisions within each region. However, decision trees can be inconvenient to deploy in standard neural inference pipelines and are not directly compatible with the ecosystem of tools developed for NN verification. A reverse transformation is therefore desirable to move between interpretable and neural models while preserving their behavior. We address this problem by focusing on the transformation from decision trees to NNs. We first discuss a constructive exact transformation from decision trees to rectified linear unit (ReLU) NNs, showing that every decision tree admits an equivalent neural realization, but also that exactness can cause substantial growth in the size of the resulting neural model. We address this issue by introducing a δ-approximate transformation method that maps a decision tree to a fixed-depth NN while relaxing exact equivalence. Specifically, our approximation guarantees exact agreement of the neural model with the decision tree outside a δ-dependent neighborhood, thereby confining possible prediction discrepancies to this neighborhood. We evaluate the transformation on neural control policies on two benchmarks from the OpenAI Gym environment. Across policies obtained from multiple NN architectures and independently trained models, we evaluate the fidelity of the converted NN with respect to its source decision tree, closed-loop performance, and preservation of formally verified properties. We further evaluate a combined pruning-and-conversion pipeline, demonstrating a tunable trade-off between policy fidelity and the size of the resulting network. Overall, our method enables a practical workflow in which policies can be analyzed and modified as interpretable decision trees and then deployed as compact NNs.

1 INTRODUCTION

Neural networks (NNs) are widely used as decision-making models ranging from perception to control tasks. Despite achieving high performance, their decision logic is often difficult to inspect, modify, and formally analyze, motivating transformations into more interpretable representations. Decision trees (DTs) are a particularly attractive alternative since they explicitly expose the partition of the input space underlying a decision policy Aytekin (2022).

Most existing work has focused on transforming NNs into DTs. Tree-extraction and distillation-based approaches learn DTs that approximate the behavior of trained networks Craven & Shavlik (1995); Frosst & Hinton (2017), while exact transformation methods construct trees that preserve the network's decision function Nguyen et al. (2020); Chang et al. (2026). However, tree representations are most useful when they support a bidirectional workflow. A NN can be transformed into a DT for inspection or modification, while deployment can be done using a neural implementation compatible with existing acceleration and verification tools Shriver et al. (2021); Wang et al. (2021). This motivates the problem we address in this work: *how can a decision tree be systematically converted back into a neural network while controlling the behavioral distortion introduced by the conversion?*

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.

DT-to-NN mappings have been explored, mainly constructing or initializing neural models that are then trained Sethi (1990); Ivanova & Kubat (1995); Setiono & Leow (1999); Humbird et al. (2019). In contrast, we seek a training-free transformation that directly constructs the network parameters from a given DT while formally characterizing the relation between the decisions of the two models.

We first consider a DT-to-NN transformation that matches the DT decision away from its split boundaries. Its main limitation is the class-wise maximum over leaf scores, whose ReLU realization requires additional reduction layers as the number of leaves grows. To avoid this dependence on tree size in the network depth, we introduce a δ-approximate transformation that replaces the class-wise maximum with the aggregation of leaves whose path-constraint violation is smaller than $\delta > 0$. The resulting NN has a fixed depth of four layers. The approximation guarantees that for every input x, the transformed NN predicts the same label as the source DT at some input x' satisfying $\|x' - x\|_2 < \kappa_S \delta$, where κ_S is a tree-dependent constant determined by its affine predicates. Thus, disagreement can occur only within a $\kappa_S \delta$-dependent neighborhood of the DT decision boundaries.

We evaluate the proposed transformation on neural control policies for the OpenAI Gym MountainCar and CartPole environments Brockman et al. (2016). We first study, for a fixed tree, how δ affects the preservation of formally verified properties, thereby isolating the effect of the approximate conversion. We then evaluate an end-to-end pruning-and-conversion workflow where increasing δ enables more aggressive pruning and consequently smaller trees and NNs. This allows us to study the trade-off between model size, policy fidelity, and closed-loop performance.

Our main contributions are:

- We provide a constructive, training-free transformation from DTs with affine predicates to ReLU NNs and characterize its architecture and complexity.
- We introduce a δ-approximate transformation that removes the class-wise maximum of the exact construction and produces a fixed-depth ReLU network.
- We provide an approximation guarantee showing that disagreement with the source DT is limited to a $\kappa_S \delta$-dependent neighborhood of its decision boundaries.
- We evaluate the transformation across multiple controller architectures, investigating the preservation of formally verified properties and the trade-off among model size, policy fidelity, and closed-loop performance in a pruning-and-conversion workflow.

2 Related Work

Neural networks to decision trees. Transformations of NNs into DTs have been extensively studied as a way to obtain more interpretable representations from learned models. Tree-extraction methods, such as TREPAN Craven & Shavlik (1995), construct a DT that approximates the predictions of a trained network. Similar approaches use distillation to transfer the behavior of a neural model into a tree Frosst & Hinton (2017). In general, these approaches aim at achieving high empirical fidelity on a set of inputs, without formally requiring functional equivalence between the source network and the extracted tree. A complementary research direction studies exact transformations. Nguyen et al. (2020) construct multi-class multivariate DTs that exactly represent the behavior of ReLU NNs. Moreover, Chang et al. (2026) construct DT representations of ReLU neural controllers while pruning redundant regions during the transformation. Such exact transformations are especially relevant to our setting as they allow the representation of neural policy as a DT without introducing any initial approximation error. In this work, we focus on the reverse direction, i.e., starting from a DT, we construct a neural representation.

Decision trees to neural networks. Mappings from DTs to NNs have a long history. Sethi (1990) introduced entropy networks, in which a DT determines the structure of a layered NN that can subsequently be trained to improve generalization. Similarly, Ivanova & Kubat (1995) use DTs to initialize NNs, while Setiono & Leow (1999) study mappings between the two model classes together with network pruning. More recently, DJINN Humbird et al. (2019) uses DTs and tree ensembles to determine the architecture and initialization of deep networks that are then further trained. However, these previous works mainly use the tree as an initialization for subsequent optimization, while our goal is to obtain a deterministic and training-free conversion in which the parameters of the NN are constructed directly from the predicates of a given DT.

3 PRELIMINARIES

For a matrix A, $A^\top$ denotes its transpose and $A_{i,j}$ its (i,j) entry; for a vector a, a_i and $(a)_i$ denote its i-th entry. Unless otherwise stated, $\|\cdot\|$ is the Euclidean norm, and $\alpha(z) = \max\{z, 0\}$ denotes the ReLU, applied element-wise. We consider a classification setting, and whenever an $\arg\max$ is not unique, a deterministic tie-breaking rule is used for both NNs and DTs.

Neural Networks. Let $\mathcal{N}$ be a feedforward ReLU NN with L layers and let W^ℓ and B^ℓ denote the weight matrix and bias vector of layer $\ell \geq 2$. For an input $x \in \mathbb{R}^n$, the activations are $h^1(x) = x$ and $h^\ell(x) = \alpha\big(W^\ell h^{\ell-1}(x) + B^\ell\big)$ for $\ell = 2, \ldots, L-1$. The output layer produces the score vector $y_\mathcal{N}(x) = W^L h^{L-1}(x) + B^L$. The network's decision function is

$$f_\mathcal{N}(x) = \arg\max_k \left(y_\mathcal{N}(x)\right)_k . \tag{1}$$

Decision Trees. Following previous work from Chang et al. (2026), we consider DTs whose internal nodes are defined by affine predicates over the input space. Let $\mathcal{S}$ be a DT with input dimension n, and let $\mathcal{I}_\mathcal{S}$ and $\mathcal{L}_\mathcal{S}$ denote its sets of internal and leaf nodes. Each internal node $v \in \mathcal{I}_\mathcal{S}$ is associated to a weight vector $w^v \in \mathbb{R}^n$ and a bias $b^v \in \mathbb{R}$, defining the affine predicate

$$p^v(x) := x^\top w^v + b^v. \tag{2}$$

Inputs with $p^v(x) \leq 0$ are routed to the left child and inputs with $p^v(x) > 0$ to the right child. Each leaf $v \in \mathcal{L}_\mathcal{S}$ has a vector $Q^v \in \mathbb{R}^c$ over the c output classes and the label $\ell(v) := \arg\max_k Q^v_k$. Starting from the root v_0, an input x is routed until it reaches a leaf v. The decision function of the tree is then defined as $f\mathcal{S}(x) := \ell(v)$.

Exact NN-to-DT transformation. In contrast to tree extraction methods, which learn a DT that approximates a trained network (Frosst & Hinton, 2017), exact transformations construct a DT that is functionally equivalent to the NN. We illustrate the transformation proposed by Chang et al. (2026) using the example in Figure 1. The NN $\mathcal{N}$ in Figure 1a has a single hidden neuron $\alpha(x_2)$ and induces three regions of the input space, $x_2 \leq 0$, $0 < x_2 \leq 0.001$, and $x_2 > 0.001$, which correspond to the leaves v_1, v_4, and v_3 of the DT $\mathcal{S}_\mathcal{N}$ in Figure 1b.

Two of these regions share the same label, together forming a single decision region. The transformation guarantees pointwise equality of the corresponding characteristic functions, i.e., $f_\mathcal{N}(x) = f_{\mathcal{S}_\mathcal{N}}(x), \forall x \in \mathbb{R}^2$ (Theorem 1 Chang et al. (2026)).

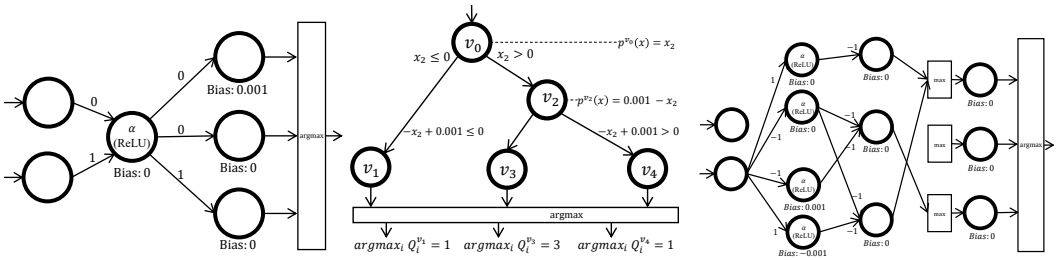
4 EXACT TRANSFORMATION FROM DECISION TREES TO NEURAL NETWORKS

After transforming the reference network $\mathcal{N}$ into a tree $\mathcal{S}_\mathcal{N}$, the tree can be inspected or modified or the reference model itself can be available as a DT. A DT-to-NN transformation becomes relevant, providing a neural model that can exploit the optimization, deployment, and verification ecosystem available for NNs. Since different architectures and parameterizations can implement the same classifier, this transformation is not merely the inverse of the NN-to-DT construction. Therefore, we construct a NN $\mathcal{N}'$ directly from the predicates and leaf labels of a DT $\mathcal{S}$, as illustrated in Figure 1c.

The construction proceeds in three stages. The first hidden layer encodes, for each internal node, the violation of the two associated branch conditions. The second hidden layer is linear and computes the negative sum of these violations along each root-to-leaf path. The leaf scores are then aggregated class-wise through a $\max$ operation. In Figure 1c, these stages correspond to the first hidden layer (one pair of neurons for each of the nodes v_0 and v_2 of Figure 1b), the second hidden layer (one neuron for each of the leaves v_1, v_3, and v_4), and the $\max$ blocks. We now formalize these rules and illustrate each of them on this example. To trace the computation, we use the input $\bar{x}$ with $\bar{x}_2 = 0.0005$, which the tree routes right at v_0 and right at v_2 to the leaf v_4, so that $f_\mathcal{S}(\bar{x}) = 1$.

4.1 CONSTRUCTION RULES

Let $\mathcal{S}$ be a DT with internal-node set $\mathcal{I}_\mathcal{S}$ and leaf set $\mathcal{L}_\mathcal{S}$. We construct a ReLU NN $\mathcal{N}'$ from $\mathcal{S}$ according to the transformation procedure described in the following.



(a) Reference NN $\mathcal{N}$. (b) Equivalent DT $\mathcal{S}_{\mathcal{N}}$. (c) Equivalent NN $\mathcal{N}'$.

Figure 1: Example of NN-to-DT-to-NN exact transformation. (a) Reference NN $\mathcal{N}$. (b) Equivalent DT $\mathcal{S}_{\mathcal{N}}$ for the NN in 1a. (c) Equivalent NN $\mathcal{N}'$ for the DT in 1b.

Predicate layer. For each internal node $u \in \mathcal{I}_{\mathcal{S}}$, we introduce two ReLU neurons

$$h_u^+(x) = \alpha(p^u(x)), \qquad h_u^-(x) = \alpha(-p^u(x)). \tag{3}$$

The activation $h_u^+(x)$ is positive when the left-branch condition $p^u(x) \le 0$ is violated, and $h_u^-(x)$ is positive when the right-branch condition $p^u(x) > 0$ is violated. At a boundary point $p^u(x) = 0$, both activations are zero. This is the source of possible ties between adjacent leaves.

In the example, $p^{v_0}(x) = x_2$ and $p^{v_2}(x) = 0.001 - x_2$ yield the four neurons $h_{v_0}^+ = \alpha(x_2)$, $h_{v_0}^- = \alpha(-x_2)$, $h_{v_2}^+ = \alpha(0.001 - x_2)$, and $h_{v_2}^- = \alpha(x_2 - 0.001)$, which form the first hidden layer in Figure 1c. At $\bar{x}$, the values of these neurons are $(0.0005,\ 0,\ 0.0005,\ 0)$. In this case, the left-branch conditions of both nodes are violated, while both right-branch conditions hold.

Leaf-score layer. The predicate neurons are then combined along each root-to-leaf path. For each leaf $v \in \mathcal{L}_{\mathcal{S}}$, let $P_L(v)$ and $P_R(v)$ denote the ancestors of v at which the path to v takes the left and right branch, respectively. Define

$$r_v(x) = -\sum_{u \in P_L(v)} h_u^+(x) - \sum_{u \in P_R(v)} h_u^-(x). \tag{4}$$

This layer has no activation function: each leaf score uses weight -1 on the predicate neurons along its path and bias 0. Since all predicate activations are nonnegative, $r_v(x) \le 0$, and $r_v(x) = 0$ whenever all predicates on the path to v are satisfied in the non-strict sense encoded by the predicate layer. Therefore, $r_v(x)$ is maximum for the leaves consistent with x.

In Figure 1b, the path to v_1 takes the left branch at v_0, the path to v_3 takes the right branch at v_0 and the left branch at v_2. The path to v_4 takes the right branch at both nodes. The corresponding leaf scores are $r_{v_1} = -h_{v_0}^+$, $r_{v_3} = -h_{v_0}^- - h_{v_2}^+$, $r_{v_4} = -h_{v_0}^- - h_{v_2}^-$. The three linear neurons in the second hidden layer in Figure 1c output the leaf scores r_{v_1}, r_{v_3}, and r_{v_4} directly to the $\max$ blocks. At $\bar{x}$, these scores are $r_{v_1} = -0.0005$, $r_{v_3} = -0.0005$, and $r_{v_4} = 0$, so only the reached leaf v_4 has score zero.

Class-wise aggregation. To obtain one score per class, we take the best score among the leaves with that label

$$z_k(x) = \max_{v:\ell(v)=k} \{r_v(x)\}, \tag{5}$$

and set $z_k(x) = -1$ if no leaf has label k. Therefore, the network prediction is $f_{\mathcal{N}'}(x) = \arg\max_k z_k(x)$.

In the example, leaves v_1 and v_4 share label 1, while v_3 has label 3, and no leaf has label 2. The aggregation rule sets $z_1 = \max\{r_{v_1}, r_{v_4}\}$, $z_2 = -1$, and $z_3 = r_{v_3}$. Using $\bar{x}$, the scores are $(z_1, z_2, z_3) = (0, -1, -0.0005)$, and the network selects class 1, in agreement with $f_{\mathcal{S}}(\bar{x})$.

ReLU realization of the maximum. The final step is to express Equation 5 with ReLU layers. Since all leaf scores are nonpositive, the maximum of two scores $a, b \le 0$ can be written as

$$\max(a, b) = \alpha(a - b) - \alpha(-b), \tag{6}$$

which uses two ReLU neurons. Applying this identity recursively in a balanced pairwise reduction computes all class-wise maxima in parallel within at most $\lceil \log_2 |\mathcal{L}_\mathcal{S}| \rceil$ additional ReLU layers.

In the example, each class has at most two leaves and, therefore, a single reduction stage is sufficient. For class 1, $z_1 = \alpha(r_{v_1} - r_{v_4}) - \alpha(-r_{v_4})$ with two neurons, whereas the unpaired score of class 3 is propagated as $z_3 = -\alpha(-r_{v_3})$. These neurons realize the $\max$ blocks in Figure 1c.

This construction matches the decision of $\mathcal{S}$ everywhere except potentially on the split boundaries, which we collect in the following set.

Definition 4.1 (Split-boundary set)**.** The split-boundary set of $\mathcal{S}$ is

$$\mathcal{H}_\mathcal{S} := \bigcup_{u \in \mathcal{I}_\mathcal{S}} \{x \in \mathbb{R}^n : p^u(x) = 0\}. \tag{7}$$

Lemma 4.2. *Let $\mathcal{S}$ be a binary decision tree on $\mathbb{R}^n$ with $m = |\mathcal{L}_\mathcal{S}|$ leaves, and assume that every leaf is reached by at least one input. Then the above construction procedure yields a ReLU network $\mathcal{N}'$ with at most $4 + \lceil \log_2 m \rceil$ layers and maximum hidden-layer width at most $2m$ such that*

$$f_{\mathcal{N}'}(x) = f_\mathcal{S}(x), \qquad \forall x \in \mathbb{R}^n \setminus \mathcal{H}_\mathcal{S}. \tag{8}$$

Moreover, for each $x \in \mathbb{R}^n$ and every $\varepsilon > 0$, there exists $x' \in \mathbb{R}^n$ such that $\|x' - x\|_2 < \varepsilon$ and $f_{\mathcal{N}'}(x) = f_\mathcal{S}(x')$.

The proof is given in Appendix A. Thus, even on a split boundary, the class returned by $\mathcal{N}'$ is returned by $\mathcal{S}$ at an arbitrarily close input. In the example, for $x_2 = 0.001$ both r_{v_3} and r_{v_4} disappear, so $z_1 = z_3 = 0$. The tie-breaking rule then selects class 1, whereas the tree routes this input to v_3 and returns class 3. Class 1 is nevertheless returned by the tree for $x_2 < 0.001$, arbitrarily close to the boundary. However, the price of exactness is the class-wise maximum, which adds up to $\lceil \log_2 m \rceil$ layers to the network, each with up to m neurons. In the next section, we remove these layers by introducing a δ-approximate transformation that yields a fixed-depth network.

5 δ-Approximate Transformation from Decision Tree to Neural Network

The procedure presented in Section 4 provides an exact transformation. However, the max operations raise a scalability issue. To address this limitation, instead of computing the maximum over the leaves of each class, we activate leaves whose aggregate root-to-leaf constraint violations are less than a parameter $\delta > 0$, and we sum these activations class-wise. This yields a NN $\mathcal{N}'_\delta$ with fixed depth of four layers. Although it is no longer exactly equivalent to $\mathcal{S}$, it agrees with $\mathcal{S}$ at every input farther from the decision boundaries of $\mathcal{S}$ than a geometry-dependent multiple of δ (Theorem 5.2).

5.1 Transformation Procedure

We assume that $\|w^v\|_2 > 0$ for every $v \in \mathcal{I}_\mathcal{S}$ and that every leaf is reachable, i.e.,

$$D_v := \{x \in \mathbb{R}^n : \mathcal{S} \text{ reaches } v \text{ on input } x\} \neq \varnothing, \qquad \forall v \in \mathcal{L}_\mathcal{S}. \tag{9}$$

Both assumptions are without loss of generality: a node with $w^v = 0$ always takes the same branch and can be replaced by that child subtree, and an unreachable leaf can be removed by replacing its parent with its sibling subtree.

In the following, we describe the construction procedure using the running example of Figure 1b. For the illustration, we consider $\delta = 0.002$ and denote by $\bar{x}$ an input with $\bar{x}_2 = 0.0005$, which the tree routes to v_4 (class 1) as in Section 4, and by $\hat{x}$ an input with $\hat{x}_2 = 0.0012$, which lies slightly above the decision boundary $x_2 = 0.001$ and is routed to v_3 (class 3).

Layer 2. For each internal node $u \in \mathcal{I}_\mathcal{S}$, we introduce the ReLU neurons

$$z_u^L(x) = \alpha\left(\frac{p^u(x)}{\|w^u\|_2}\right), \qquad z_u^R(x) = \alpha\left(-\frac{p^u(x)}{\|w^u\|_2}\right). \tag{10}$$

Because of the normalization, $z_u^L(x)$ and $z_u^R(x)$ are the Euclidean distances from x to the half-spaces $\{p^u \leq 0\}$ and $\{p^u \geq 0\}$.

In the example, $\|w^{v_0}\|_2 = \|w^{v_2}\|_2 = 1$, so Layer 2 coincides with the predicate layer of Section 4: $z^L_{v_0} = \alpha(x_2)$, $z^R_{v_0} = \alpha(-x_2)$, $z^L_{v_2} = \alpha(0.001 - x_2)$, and $z^R_{v_2} = \alpha(x_2 - 0.001)$.

Layer 3. With $P_L(v)$ and $P_R(v)$ as in Section 4, the aggregate path violation of a leaf v is

$$V_v(x) := \sum_{u \in P_L(v)} z^L_u(x) + \sum_{u \in P_R(v)} z^R_u(x) \ \geq 0, \tag{11}$$

and $V_v(x) = 0$ if and only if x lies in the closed polyhedron

$$P_v := \big\{ z : p^u(z) \leq 0 \ \forall u \in P_L(v), \ \ p^u(z) \geq 0 \ \forall u \in P_R(v) \big\}, \tag{12}$$

which is the closure of D_v under assumption Equation 9 (Lemma A.1 in Appendix A). For each leaf v, we introduce the ReLU neuron

$$h_v(x) = \alpha(\delta - V_v(x)), \qquad \text{so that} \qquad h_v(x) > 0 \iff V_v(x) < \delta. \tag{13}$$

Layers 2 and 3 are the normalized counterparts of Equations 3and 4, with an additional ReLU and bias δ at the leaves. The bias is essential, since for $\delta = 0$ all leaf neurons would vanish identically. If v_x is the leaf reached by $\mathcal{S}$ on input x, then $V_{v_x}(x) = 0$ and hence $h_{v_x}(x) = \delta$.

In our example, the aggregate violations are

$$V_{v_1} = z^L_{v_0}, \qquad V_{v_3} = z^R_{v_0} + z^L_{v_2}, \qquad V_{v_4} = z^R_{v_0} + z^R_{v_2}, \tag{14}$$

and the leaf polyhedra are $P_{v_1} = \{x_2 \leq 0\}$, $P_{v_3} = \{x_2 \geq 0.001\}$, and $P_{v_4} = \{0 \leq x_2 \leq 0.001\}$. At $\bar{x}$, we obtain $(V_{v_1}, V_{v_3}, V_{v_4}) = (0.0005,\, 0.0005,\, 0)$ and hence $(h_{v_1}, h_{v_3}, h_{v_4}) = (0.0015,\, 0.0015,\, 0.002)$. Unlike in the exact construction, the leaves v_1 and v_3, which are not reached, are also active, because their violations are smaller than δ.

Layer 4. The output layer sums the activations of the leaves of each class:

$$\big(y_{\mathcal{N}'_\delta}(x)\big)_k = \sum_{v:\, \ell(v)=k} h_v(x), \qquad f_{\mathcal{N}'_\delta}(x) = \arg\max_k \big(y_{\mathcal{N}'_\delta}(x)\big)_k. \tag{15}$$

Compared with Section 4, the $\max$ is now replaced by a sum, so no reduction layers are needed.

In our example, the class scores are $y_1 = h_{v_1} + h_{v_4}$, $y_2 = 0$, and $y_3 = h_{v_3}$. At $\bar{x}$, we obtain $(y_1, y_2, y_3) = (0.0035,\, 0,\, 0.0015)$, and the network selects class 1, in agreement with the tree. At $\hat{x}$, however, we obtain $(y_1, y_2, y_3) = (0.0026,\, 0,\, 0.002)$. The network again selects class 1, whereas the tree returns class 3. Although v_3 has the largest single activation, the two partially active class-1 leaves v_1 and v_4 jointly outvote it. This is the price of replacing the maximum by a sum. More generally, the Layer 3 activations of the example are

$$\big(h_{v_1},\, h_{v_3},\, h_{v_4}\big)^\top = \begin{cases} \big(\delta,\, \alpha(\delta + 2x_2 - 0.001),\, \alpha(\delta + x_2)\big)^\top, & x_2 \leq 0, \\ \big(\alpha(\delta - x_2),\, \alpha(\delta + x_2 - 0.001),\, \delta\big)^\top, & 0 < x_2 < 0.001, \\ \big(\alpha(\delta - x_2),\, \delta,\, \alpha(\delta - x_2 + 0.001)\big)^\top, & x_2 \geq 0.001, \end{cases} \tag{16}$$

For $x_2 < 0.001$, we have $y_1 \geq \delta > y_3$, so the network agrees with the tree. For $x_2 \geq 0.001$, the tree predicts class 3 with $y_3 = \delta$, while lowest-index tie-breaking selects class 1 exactly when $0.001 \leq x_2 \leq (\delta + 0.001)/2$, or only at $x_2 = 0.001$ if $\delta \leq 0.001$. For $\delta = 0.002$, the disagreement region is therefore $0.001 \leq x_2 \leq 0.0015$, which contains $\hat{x}$. At $x_2 = 0.002$, for instance, we have $y_1 = 0.001 < y_3 = 0.002$, and the network agrees with the tree again. In general, all disagreements lie within distance $\max\{0, (\delta - 0.001)/2\} < \delta$ of the decision boundary.

5.2 Approximation Guarantee

Since V_v sums the distances to individual half-spaces, the distance to their intersection P_v also depends on the angles between the predicate hyperplanes. Let $q_u := w^u/\|w^u\|_2$ and $P(v) = P_L(v) \cup P_R(v)$. For every $J \subseteq P(v)$ with $\{q_u\}_{u \in J}$ linearly independent and every $i \in J$, let

$$\theta_{i,J} := \angle(q_i,\, \mathrm{span}\{q_u : u \in J \setminus \{i\}\}) \in (0, \pi/2], \tag{17}$$

with $\theta_{i,\{i\}} = \pi/2$, so that $\csc \theta_{i,J} = 1/\operatorname{dist}_2(q_i, \mathrm{span}\{q_u : u \in J \setminus \{i\}\})$. We define the constant

$$\kappa_{\mathcal{S}} := \max\Big(\{1\} \cup \big\{\csc \theta_{i,J} : v \in \mathcal{L}_{\mathcal{S}},\ J \subseteq P(v),\ i \in J\big\}\Big), \tag{18}$$

where J ranges over all the linearly independent subsets. $\kappa_{\mathcal{S}}$ is a finite constant, invariant to rescaling of the weights, and not tight. For axis-aligned trees, all admissible J contain orthogonal vectors, and therefore $\kappa_{\mathcal{S}} = 1$. In our example, since $q_{v_0} = e_2$ and $q_{v_2} = -e_2$ are linearly dependent, the only admissible sets J are singletons, which again gives $\kappa_{\mathcal{S}} = 1$.

Lemma 5.1. *For every $v \in \mathcal{L}_{\mathcal{S}}$ and $x \in \mathbb{R}^n$, $\mathrm{dist}_2(x, P_v) \le \kappa_{\mathcal{S}}\, V_v(x)$.*

Theorem 5.2 (δ-Approximation Guarantee)**.** *Let $\mathcal{S}$ satisfy the assumptions of this section, let $\delta > 0$, and let $\mathcal{N}'_\delta$ be the network obtained through the δ-approximate transformation. Then, for every $x \in \mathbb{R}^n$, there exists $x' \in \mathbb{R}^n$ with*

$$\|x' - x\|_2 < \kappa_{\mathcal{S}}\, \delta \qquad \textit{and} \qquad f_{\mathcal{N}'_\delta}(x) = f_{\mathcal{S}}(x'). \tag{19}$$

In particular, if $f_{\mathcal{S}}$ is constant on $B_2(x, \kappa_{\mathcal{S}}\delta)$, then $f_{\mathcal{N}'_\delta}(x) = f_{\mathcal{S}}(x)$.

Both proofs are provided in Appendix B. Therefore, $\mathcal{N}'_\delta$ can disagree only at points lying within distance $\kappa\mathcal{S}\delta$ of a decision boundary of $\mathcal{S}$. Unlike the exact transformation, this disagreement region has positive volume in general, and it shrinks as δ decreases. In our example, the network returns class 1 at $\hat{x}$. As guaranteed by Theorem 5.2 with $\kappa_{\mathcal{S}}\delta = 0.002$, the tree also returns class 1 at inputs with $x_2 < 0.001$, which lie within distance 0.0002 of $\hat{x}$.

Lemma 5.3. *The network $\mathcal{N}'_\delta$ has four layers and $2|\mathcal{I}_{\mathcal{S}}| + |\mathcal{L}_{\mathcal{S}}| + c$ non-input neurons.*

For a DT with m leaves, we have $3m - 2 + c$ non-input neurons compared with the up to $\lceil \log_2 m \rceil$ additional layers of width up to m required by the exact transformation (Lemma 4.2). In the running example, $\mathcal{N}'_\delta$ has $2 \cdot 2 + 3 + 3 = 10$ non-input neurons, whereas the exact network in Figure 1c additionally requires the layer that implements the $\max$ blocks. The architecture is independent of δ, which only sets the biases of Layer 3. Thus, δ controls the approximation error, and the network size depends on the tree.

6 Case Studies

We evaluate the proposed δ-approximate transformation on two standard OpenAI Gym environments (Brockman et al., 2016). The code used for our experiments is released as part of the artifact.[1] Our experiments are guided by the following research questions.

- **RQ1:** For a fixed DT, how does δ affect the preservation of properties formally verified on the reference model? (Section 6.1)
- **RQ2:** In a workflow in which the DT is pruned before conversion, is there a trade-off between the size of the resulting network, its fidelity to the original policy, and its closed-loop performance? (Section 6.2)

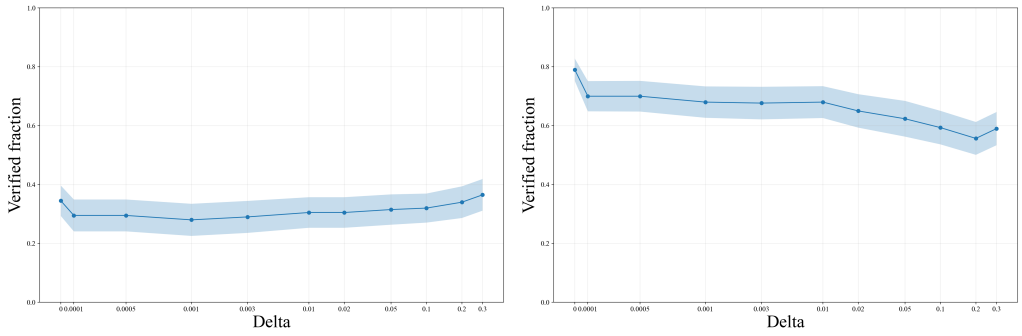
Experimental Setup We use the MountainCar-v0 (Moore, 1990) and CartPole-v1 (Barto et al., 1983) environments from OpenAI Gym (Brockman et al., 2016), described in Appendix C. For each environment, we train reference controllers $\mathcal{N}$ with DQN (Mnih et al., 2013) using five architectures and 20 random seeds per architecture, i.e., 100 controllers per environment (Appendix E). For each model, we then use the exact NN-to-DT transformation of Chang et al. (2026) to obtain a DT $\mathcal{S}_{\mathcal{N}}$ equivalent to $\mathcal{N}$, which serves as the reference DT.

6.1 Preservation of Verified Properties (RQ1)

Theorem 5.2 characterizes the region in which the transformed network may disagree with its source tree in terms of δ and $\kappa_{\mathcal{S}}$. We complement this result by studying how δ affects properties verified on the reference model. To isolate the effect of the conversion, we use a single tree per controller and convert the resulting tree for different values of δ, obtaining multiple NNs $\mathcal{N}'_\delta$.

For MountainCar, we verify *local action consistency* around a nominal state on the left slope and the absence of the left action near the goal. For CartPole, we verify local action consistency around

[1] `https://anonymous.4open.science/r/sdt_to_nn-B215`. The experiments were conducted on a 13th Gen Intel Core i9-13980HX with an RTX 4070 GPU.



(a) MountainCar. (b) CartPole.

Figure 2: Verified fraction as function of δ, results are averaged over architectures and seeds, with 95% confidence intervals. $\delta = 0$ is used for the reference baseline evaluated on $\mathcal{N}$.

a nominal state and two properties requiring the cart to be pushed toward the side to which the pole leans when it is close to instability (Appendix D). Each property is checked with DNNV (Shriver et al., 2021) using nnenum (Bak, 2021) and counts as verified only when it is proven. The *verified fraction* of a network is the fraction of verified properties. We use $\delta = 0$ to denote the reference baseline, for which we verify $\mathcal{N}$ itself, since it is equivalent to $\mathcal{S}_{\mathcal{N}}$ (Chang et al., 2026).

Figure 2 reports the results. For completeness, we report the verified fraction for each architecture separately in Appendix F. Since the reference controllers are trained for reward and not for these properties, they satisfy only some of them, and preservation should be read relative to the baseline.

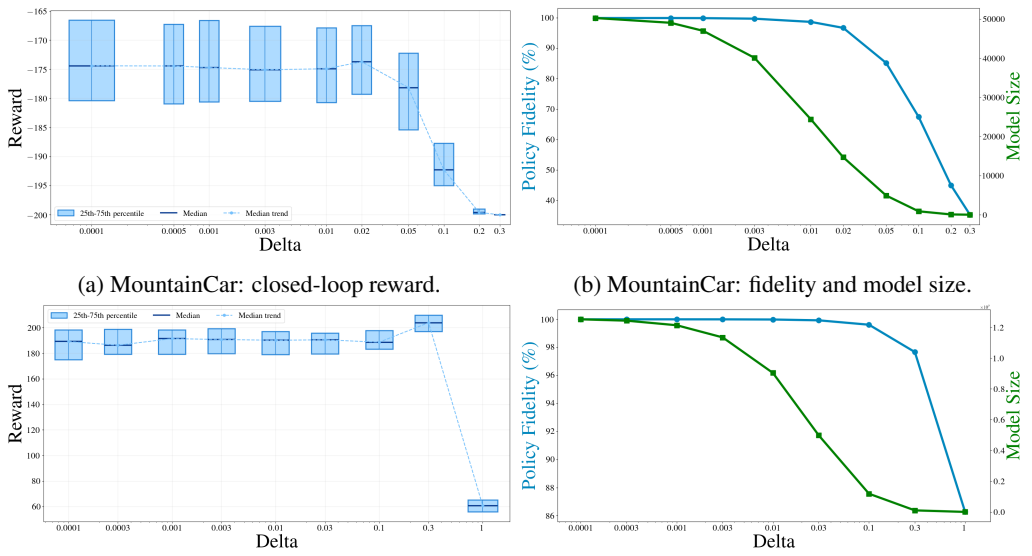
In both environments, the largest change occurs when moving from the baseline to the approximate construction. For small and moderate values of δ, the verified fraction then remains nearly constant, indicating that the conversion preserves most of the properties. The sensitivity to larger values of δ differs across the two tasks, as it depends on the tree geometry through $\kappa_{\mathcal{S}}$, on the scale of the state variables, and on the regions covered by the properties. Inconclusive verifier results, about 8% of the checks for MountainCar, are counted as not verified.

6.2 Size, Fidelity, and Closed-Loop Performance (RQ2)

In this experiment, we evaluate a practical pruning-conversion pipeline. In particular, each reference tree $\mathcal{S}_{\mathcal{N}}$ is pruned by removing the branches whose region is too narrow to contain a point at a margin of at least δ from its splitting hyperplanes. The pruned tree $\mathcal{S}_\delta$ is then converted into $\mathcal{N}'_\delta$ with the same δ. Intuitively, pruning discards thin regions near decision boundaries, where the δ-approximate transformation may deviate from the tree anyway. By Lemma 5.3, pruning reduces the size of the network, highlighting an interesting trade-off between the model size of $\mathcal{N}'_\delta$ and fidelity to the reference tree model. While RQ1 investigates the effect of δ on preserving previously verified properties, RQ2 discusses a potential benefit of using larger approximation levels.

For each $\mathcal{N}'_\delta$, we report the closed-loop reward over 100 episodes with random initial states and horizons of 200 (MountainCar) and 500 (CartPole) steps, the *end-to-end fidelity*, i.e., the action agreement with $\mathcal{N}$ (equivalently, $\mathcal{S}_{\mathcal{N}}$) over the state space, and the number of parameters.

Figure 3 shows the results of this evaluation. In particular, it is possible to see that increasing δ produces smaller trees and networks while gradually reducing end-to-end fidelity. For MountainCar, the network size decreases substantially while fidelity remains close to 100%. For CartPole, the network size decreases from about 1.2×10^7 parameters at $\delta = 10^{-4}$, reflecting the complexity of the corresponding trees, to a small fraction of this value, while fidelity remains above 97% up to $\delta = 0.3$. The closed-loop reward remains comparatively stable over a broad range of δ and degrades mainly at the largest tested values. Hence, a reduction in global action agreement does not necessarily translate into a proportional loss of task-level performance, since disagreements may occur in states that are rarely visited in closed loop or multiple actions might be equally good. We do not interpret small non-monotonic variations, such as the reward increase for CartPole around



(a) MountainCar: closed-loop reward.

(b) MountainCar: fidelity and model size.

(c) CartPole: closed-loop reward.

(d) CartPole: fidelity and model size.

Figure 3: Results of the pruning-and-conversion pipeline versus δ, which is used both as the pruning margin and as the parameter of the subsequent DT-to-NN conversion. The smallest tested value is $\delta = 10^{-4}$. The reduction in network size results from the smaller pruned trees.

$\delta = 0.3$, as a systematic improvement. Overall, pruning controlled by δ yields a tunable trade-off between model size, fidelity to the original policy, and closed-loop performance.

7 Conclusion

In this work, we address the problem of converting a decision tree into a neural representation while controlling the behavioral distortion introduced by the transformation. We first presented a constructive ReLU realization of a DT and characterized the complexity introduced by the class-wise maximum over leaf scores. We then introduced a δ-approximate transformation that removes this operation and produces a four-layer ReLU network.

The proposed approximation guarantees geometrically that the transformed network can disagree with its source DT only within a $\kappa_S\delta$-dependent neighborhood of its decision boundaries, where κ_S captures the geometry of the tree predicates. Experiments on MountainCar and CartPole show that, for fixed trees, moderate approximation levels preserve most of the properties that were verified for the source tree. We then show a pruning-and-conversion workflow, where increasing δ enables smaller tree and network representations while maintaining policy fidelity and closed-loop performance over a broad range of values. These results support the use of DTs as intermediate representations that can be inspected or modified and then converted into NNs for deployment. Overall, this work supports a practical workflow in which DTs serve as intermediate representations that can be analyzed or edited and then converted back into compact neural controllers.

Limitations and future work. The current method uses a uniform δ across the input space, which may be suboptimal in regions with different sensitivities to boundary perturbations. Furthermore, the approximation guarantee depends on the tree geometry through κ_S, which may become large for unfavorable configurations of affine predicates. Future work will investigate higher-dimensional tasks, tighter or locally adaptive geometric bounds, and adaptive target network architectures whose depth and width are selected according to the structure and complexity of the source tree, rather than being fixed a priori. Furthermore, we plan to explore verification-aware approximation strategies that account explicitly for the properties to be preserved during transformation.

AI USE STATEMENT

In this work, we used generative AI tools for text editing, improving the clarity and readability of the manuscript. We have not used generative AI tools for tasks requiring mandatory disclosure, including the development of the methodology, formulation of hypotheses, derivation of mathematical results, design or execution of experiments, or interpretation of experimental results, and data collection or cleaning are not applicable to this work. All technical content, mathematical statements and proofs, experimental results, and cited references were independently reviewed and verified by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the definitions, constructions, and algorithms required to reproduce the proposed approach in the main text and Appendix. Detailed proofs of the theoretical results are included in the appendix. We also report the experimental setup, datasets, model configurations, hyperparameters, and evaluation procedures used in our experiments. The code used to run the experiments is provided with the submission.

REFERENCES

Caglar Aytekin. Neural Networks are Decision Trees. *arXiv preprint arXiv:2210.05189*, 2022.

Stanley Bak. nnenum: Verification of ReLU neural networks with optimized abstraction refinement. In *NASA Formal Methods (NFM)*, volume 12673 of *Lecture Notes in Computer Science*, pp. 19–36. Springer, 2021.

Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 1983.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai Gym. *arXiv preprint arXiv:1606.01540*, 2016.

Kevin Chang, Nathan Dahlin, Rahul Jain, and Pierluigi Nuzzo. Equivalent and compact representations of neural network controllers with decision trees. *IEEE Transactions on Automatic Control*, 2026.

Mark Craven and Jude Shavlik. Extracting tree-structured representations of trained networks. *Advances in neural information processing systems*, 8, 1995.

Nicholas Frosst and Geoffrey Hinton. Distilling a Neural Network Into a Soft Decision Tree. *arXiv preprint arXiv:1711.09784*, 2017.

Kelli D. Humbird, J. Luc Peterson, and Ryan G. McClarren. Deep neural network initialization with decision trees. *IEEE Transactions on Neural Networks and Learning Systems*, 30(5):1286–1295, 2019. doi: 10.1109/TNNLS.2018.2869694.

Irena Ivanova and Miroslav Kubat. Initialization of neural networks by means of decision trees. *Knowledge-Based Systems*, 8(6):333–344, 1995.

Yogesh Kumar and Pankaj Kumar. Empirical study of deep reinforcement learning algorithms for CartPole problem. In *2024 11th International Conference on Signal Processing and Integrated Networks (SPIN)*, pp. 78–83. IEEE, 2024.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602*, 2013.

Andrew William Moore. Efficient Memory-based Learning for Robot Control. Technical report, University of Cambridge, Computer Laboratory, 1990.

Duy T Nguyen, Kathryn E Kasmarik, and Hussein A Abbass. Towards Interpretable ANNs: An Exact Transformation to Multi-Class Multivariate Decision Trees. *arXiv preprint arXiv:2003.04675*, 2020.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

Ishwar K. Sethi. Entropy nets: From decision trees to neural networks. *Proceedings of the IEEE*, 78 (10):1605–1613, 1990. doi: 10.1109/5.58346.

Rudy Setiono and Wee Kheng Leow. On mapping decision trees and neural networks. *Knowledge-Based Systems*, 12(3):95–99, 1999. doi: 10.1016/S0950-7051(99)00009-X.

David Shriver, Sebastian Elbaum, and Matthew B. Dwyer. DNNV: A framework for deep neural network verification. In *Computer Aided Verification (CAV)*, volume 12759 of *Lecture Notes in Computer Science*, pp. 137–150. Springer, 2021.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Neural Network Robustness Verification. *Advances in Neural Information Processing Systems*, 34:29909–29921, 2021.

A Proof of Lemma 4.2

We first state an auxiliary result on the leaf regions of a decision tree, which is used both here and in Appendix B. For a leaf $v \in \mathcal{L}_{\mathcal{S}}$, let $P_L(v)$ and $P_R(v)$ be as in Section 4, and let

$$P_v = \big\{ z \in \mathbb{R}^n : p^u(z) \le 0 \ \ \forall u \in P_L(v), \ \ p^u(z) \ge 0 \ \ \forall u \in P_R(v) \big\}$$

be the closed polyhedron of Equation 12. Since the routing of an input depends only on the predicates along its path, the routed cell of v is

$$D_v = \big\{ z \in \mathbb{R}^n : p^u(z) \le 0 \ \ \forall u \in P_L(v), \ \ p^u(z) > 0 \ \ \forall u \in P_R(v) \big\}.$$

Lemma A.1 (Closure of routed cells)**.** *Let $v \in \mathcal{L}_{\mathcal{S}}$ be a leaf with $D_v \neq \varnothing$. Then $\overline{D_v} = P_v$. In particular, for every $x \in P_v$ and every $\varepsilon > 0$, there exists $x' \in D_v$ such that $\|x' - x\|_2 < \varepsilon$.*

Proof. Clearly $D_v \subseteq P_v$, and since P_v is closed, $\overline{D_v} \subseteq P_v$. Conversely, fix $x \in P_v$ and some $y \in D_v$, and define $x^{(t)} = (1-t)x + ty$ for $t \in (0, 1]$. Since every p^u is affine, $p^u(x^{(t)}) = (1-t)p^u(x) + tp^u(y)$. Hence, $p^u(x^{(t)}) \le 0$ for every $u \in P_L(v)$, and $p^u(x^{(t)}) \ge t\, p^u(y) > 0$ for every $u \in P_R(v)$. Thus, $x^{(t)} \in D_v$ for all $t \in (0, 1]$. Since $\|x^{(t)} - x\|_2 = t\|y - x\|_2$, choosing t sufficiently small yields $\|x^{(t)} - x\|_2 < \varepsilon$. This proves the second claim, and hence $P_v \subseteq \overline{D_v}$. □

We now prove Lemma 4.2.

Proof of Lemma 4.2. Let $m = |\mathcal{L}_{\mathcal{S}}|$. If $m = 1$, then $\mathcal{S}$ represents a constant function, and the claim follows immediately from a constant network. Assume henceforth that $m \ge 2$.

Realization. Since $r_v(x) \ge 0$, the path-violation neurons can be realized as ReLU neurons $\alpha(r_v(x)) = r_v(x)$, and the negation $-r_v$ can be absorbed into the weights of the first reduction stage. Moreover, the identities $\max(a, b) = \alpha(a - b) - \alpha(-b)$ and $a = -\alpha(-a)$ hold exactly for $a, b \le 0$, and all leaf scores and intermediate maxima are nonpositive. Hence, $\mathcal{N}'$ is a standard ReLU network in the sense of Section 3 whose output layer computes exactly $z_k(x)$ in Equation 5 for every class k, and $f_{\mathcal{N}'}(x) = \arg\max_k z_k(x)$.

Exact agreement off the split boundaries. Fix $x \notin \mathcal{H}_{\mathcal{S}}$, and let $v^\star$ be the leaf reached by $\mathcal{S}$ on input x. Every branch condition along the path to $v^\star$ is satisfied. Since $p^u(x) \neq 0$ for all $u \in \mathcal{I}_{\mathcal{S}}$, we have $p^u(x) < 0$ and hence $h_u^+(x) = 0$ for every $u \in P_L(v^\star)$, and $p^u(x) > 0$ and hence $h_u^-(x) = 0$ for every $u \in P_R(v^\star)$. Therefore, $r_{v^\star}(x) = 0$.

Now consider any leaf $v \neq v^\star$, and let u be the first internal node at which the paths to v and $v^\star$ differ. Since $p^u(x) \neq 0$, the branch taken towards $v^\star$ is satisfied strictly, whereas the branch taken towards v is strictly violated. Consequently, the corresponding term $h_u^+(x)$ or $h_u^-(x)$ in $r_v(x)$ is strictly positive, and $r_v(x) > 0$. Thus, $v^\star$ is the unique leaf with zero path violation.

Let $k^\star = \ell(v^\star) = f_{\mathcal{S}}(x)$. Then $z_{k^\star}(x) = 0$, whereas $z_k(x) < 0$ for every class $k \neq k^\star$, either because all leaves labeled k have positive path violation or because $z_k(x) = -1$. Thus, $k^\star$ is the unique maximizing class, and $f_{\mathcal{N}'}(x) = k^\star = f_{\mathcal{S}}(x)$. This proves exact agreement on $\mathbb{R}^n \setminus \mathcal{H}_{\mathcal{S}}$.

Agreement up to arbitrarily small perturbations. Fix an arbitrary $x \in \mathbb{R}^n$ and $\varepsilon > 0$. The leaf reached by $\mathcal{S}$ on input x satisfies all branch conditions along its path in the non-strict sense and therefore has zero path violation. Since $r_v(x) \geq 0$ for all leaves, it follows that $\max_k z_k(x) = 0$. Let $k = f_{\mathcal{N}'}(x)$. Since k maximizes $z_k(x)$, we have $z_k(x) = 0$, and by Equation 5 there exists a leaf v with $\ell(v) = k$ and $r_v(x) = 0$. Since every term of $r_v(x)$ vanishes, $p^u(x) \leq 0$ for all $u \in P_L(v)$ and $p^u(x) \geq 0$ for all $u \in P_R(v)$, i.e., $x \in P_v$. By assumption, $D_v \neq \varnothing$, so Lemma A.1 yields $x' \in D_v$ with $\|x' - x\|_2 < \varepsilon$. Since $\mathcal{S}$ reaches v on input x', we obtain $f_{\mathcal{S}}(x') = \ell(v) = k = f_{\mathcal{N}'}(x)$.

Size. The predicate layer contains two neurons for each internal node. Since a binary tree with m leaves has $m-1$ internal nodes, this layer has width $2(m-1)$. The path-violation layer contains one neuron for each leaf and therefore has width m. The class-wise maxima are computed by a balanced pairwise reduction. If $s \leq m$ scores remain at a given reduction stage, each pair requires two ReLU neurons and each unpaired or already completed score requires one ReLU neuron for propagation. Hence, each reduction stage has at most $s \leq m$ neurons. Since no class is associated with more than m leaves, at most $\lceil \log_2 m \rceil$ reduction stages are required, after which the linear output layer returns $z_1(x), \ldots, z_c(x)$.

Counting the input layer, the predicate layer, the path-violation layer, the reduction layers, and the output layer, the network has at most $4 + \lceil \log_2 m \rceil$ layers. Finally, every hidden layer has width at most $\max\{2(m-1), m\} \leq 2m$. □

B Proofs of Lemma 5.1 and Theorem 5.2

Throughout this appendix, we use the assumptions of Section 5, i.e., $\|w^u\|_2 > 0$ for all $u \in \mathcal{I}_{\mathcal{S}}$ and $D_v \neq \varnothing$ for all $v \in \mathcal{L}_{\mathcal{S}}$.

Signed constraint representation. For a leaf v and an internal node $u \in P(v) = P_L(v) \cup P_R(v)$, let $s_u^v = 1$ if $u \in P_L(v)$ and $s_u^v = -1$ if $u \in P_R(v)$, and define

$$a_u^v = s_u^v \, q_u = s_u^v \frac{w^u}{\|w^u\|_2}, \qquad \beta_u^v = -s_u^v \frac{b^u}{\|w^u\|_2}.$$

Then $(a_u^v)^\top x - \beta_u^v = s_u^v \, p^u(x)/\|w^u\|_2$. Stacking these rows and thresholds into A_v and β_v, we obtain

$$P_v = \{z \in \mathbb{R}^n : A_v z \leq \beta_v\},$$

and, by Eqs. 10 and 11,

$$V_v(x) = \sum_{u \in P(v)} \left((a_u^v)^\top x - \beta_u^v\right)_+ = \left\|(A_v x - \beta_v)_+\right\|_1, \tag{20}$$

where $(\cdot)_+$ denotes the entrywise positive part. In particular, $V_v(x) = 0$ if and only if $x \in P_v$. Moreover, $P_v \supseteq D_v \neq \varnothing$.

Lemma B.1. *Let $v \in \mathcal{L}_{\mathcal{S}}$. Then, for every $x \in \mathbb{R}^n$,*

$$\operatorname{dist}_2(x, P_v) \leq \kappa_{\mathcal{S}} \left\|(A_v x - \beta_v)_+\right\|_1.$$

Proof. Since P_v is a nonempty closed convex set, the Euclidean projection π of x onto P_v exists. Set $d = x - \pi$, so that $\operatorname{dist}_2(x, P_v) = \|d\|_2$. If $d = 0$, the assertion is immediate. Otherwise, by the optimality condition of the projection, d lies in the normal cone of P_v at π, which is generated by the rows a_u^v of the constraints active at π. By Carathéodory's theorem for cones, we can write $d = A_J^\top \lambda$ with $\lambda \geq 0$, where $J \subseteq P(v)$ indexes active constraints whose rows $\{a_u^v\}_{u \in J}$ are

linearly independent. Indeed, a representation with minimal support has linearly independent rows, since a linear dependence would permit changing the coefficients until one of them vanishes while preserving nonnegativity.

For each $i \in J$, define

$$g_i = A_J^\top (A_J A_J^\top)^{-1} e_i,$$

where e_i is the coordinate vector indexed by i in $\mathbb{R}^J$. Then $g_i^\top d = e_i^\top (A_J A_J^\top)^{-1} A_J A_J^\top \lambda = \lambda_i$. Moreover, g_i lies in the span of $\{a_u^v\}_{u \in J}$ and satisfies $(a_j^v)^\top g_i = 0$ for $j \in J \setminus \{i\}$ and $(a_i^v)^\top g_i = 1$. Hence, $g_i = \rho_i / \|\rho_i\|_2^2$, where ρ_i is the component of a_i^v orthogonal to $\operatorname{span}\{a_j^v : j \in J \setminus \{i\}\}$. Since $\|a_i^v\|_2 = 1$, we have $\|\rho_i\|_2 = \sin\theta_{i,J}$, where the angle is unaffected by the signs s_u^v. With the convention $\theta_{i,\{i\}} = \pi/2$ for singletons, Equation 18 gives

$$\|g_i\|_2 = \csc\theta_{i,J} \le \kappa_{\mathcal{S}}.$$

Consequently, $\lambda_i = g_i^\top d \le \kappa_{\mathcal{S}} \|d\|_2$ for every $i \in J$. Since the constraints in J are active at π, i.e., $A_J \pi = \beta_J$, we obtain

$$\begin{aligned} \|d\|_2^2 = \lambda^\top A_J d &= \sum_{i \in J} \lambda_i \big((a_i^v)^\top x - \beta_i^v\big) \\ &\le \sum_{i \in J} \lambda_i \big((a_i^v)^\top x - \beta_i^v\big)_+ \\ &\le \kappa_{\mathcal{S}} \|d\|_2 \, \big\|(A_v x - \beta_v)_+\big\|_1. \end{aligned}$$

Dividing by $\|d\|_2 > 0$ proves the claim. □

Proof of Lemma 5.1. The claim follows directly from Lemma B.1 and Equation 20. □

Proof of Theorem 5.2. Fix $x \in \mathbb{R}^n$, and let v_x be the leaf reached by $\mathcal{S}$ on input x. Since $x \in D_{v_x} \subseteq P_{v_x}$, we have $V_{v_x}(x) = 0$ and hence $h_{v_x}(x) = \delta$. By Equation 15, the output coordinate indexed by $\ell(v_x) = f_{\mathcal{S}}(x)$ contains the summand $h_{v_x}(x)$, and all summands are nonnegative. Therefore,

$$\big(y_{\mathcal{N}'_\delta}(x)\big)_{f_{\mathcal{S}}(x)} \ge \delta.$$

Let $y := f_{\mathcal{N}'_\delta}(x)$. Since y maximizes the output scores, independently of the tie-breaking rule,

$$\big(y_{\mathcal{N}'_\delta}(x)\big)_y \ge \big(y_{\mathcal{N}'_\delta}(x)\big)_{f_{\mathcal{S}}(x)} \ge \delta > 0.$$

By Equation 15, $\big(y_{\mathcal{N}'_\delta}(x)\big)_y$ is the sum of $h_v(x)$ over all leaves v with $\ell(v) = y$. Hence, there exists a leaf v' with $\ell(v') = y$ and $h_{v'}(x) > 0$, which by Equation 13 is equivalent to $V_{v'}(x) < \delta$.

By Lemma 5.1, the projection π of x onto $P_{v'}$ satisfies

$$\|\pi - x\|_2 \le \kappa_{\mathcal{S}} \, V_{v'}(x) < \kappa_{\mathcal{S}} \, \delta.$$

Let $\varepsilon = \kappa_{\mathcal{S}}\delta - \|\pi - x\|_2 > 0$. Since $\pi \in P_{v'}$ and $D_{v'} \neq \varnothing$, Lemma A.1 yields $x' \in D_{v'}$ with $\|x' - \pi\|_2 < \varepsilon$, and hence

$$\|x' - x\|_2 \le \|x' - \pi\|_2 + \|\pi - x\|_2 < \kappa_{\mathcal{S}} \, \delta.$$

Since $\mathcal{S}$ reaches v' on input x', we obtain

$$f_{\mathcal{S}}(x') = \ell(v') = y = f_{\mathcal{N}'_\delta}(x).$$

This proves the first claim.

Finally, if $f_{\mathcal{S}}$ is constant on $B_2(x, \kappa_{\mathcal{S}}\delta)$, then x' belongs to this ball, and $f_{\mathcal{N}'_\delta}(x) = f_{\mathcal{S}}(x') = f_{\mathcal{S}}(x)$. □

C Environment of Case Study

MountainCar-v0. The MountainCar task involves driving an underpowered car from a valley to a hilltop within a fixed time horizon T (Moore, 1990). The state $s_t \in \mathbb{R}^2$ comprises the car's position $s_{t,1}$ and velocity $s_{t,2}$, while the action $u_t \in \{0, 1, 2\}$ represents acceleration to the left, no acceleration, or acceleration to the right. The goal is to reach the flag as quickly as possible, with a reward of -1 per time step.

CartPole-v1. The CartPole task involves balancing a pole on a cart moving along a frictionless track for a fixed horizon T (Kumar & Kumar, 2024). The state $s_t \in \mathbb{R}^4$ consists of the cart position $s_{t,1}$, cart velocity $s_{t,2}$, pole angle $s_{t,3}$, and pole angular velocity $s_{t,4}$. The action $u_t \in \{0, 1\}$ pushes the cart to the left or to the right. A reward of 1 is awarded per step while the pole remains upright, and episodes are capped at 500 steps (Kumar & Kumar, 2024).

In both environments, the state s_t is the input $x \in \mathbb{R}^n$ of the controllers and decision trees, with $n = 2$ and $n = 4$, respectively.

D VERIFIED PROPERTIES

This section gives the formal definitions of the properties verified in Section 6.1. We state the properties for a generic controller $\mathcal{N}'_\delta$ with output score vector $y_{\mathcal{N}'_\delta}(x)$ and decision function $f_{\mathcal{N}'_\delta}(x) = \arg\max_a \big(y_{\mathcal{N}'_\delta}(x)\big)_a$, where we index the output scores by the actions of the environment and ties are broken towards the lowest index. Following Appendix C, x_1 and x_2 denote the position and velocity of the car in MountainCar, and $x_1, \ldots, x_4$ denote the cart position, cart velocity, pole angle, and pole angular velocity in CartPole.

MountainCar. We define two properties capturing local robustness and goal-directed behavior. The actions are $a \in \{0, 1, 2\}$, corresponding to acceleration to the left, no acceleration, and acceleration to the right.

(1) Local action consistency. At the nominal state $x^{(0)} = (-0.9, -0.03)$, the car is on the left slope and moving to the left. Accelerating to the left ($a = 0$) builds up the momentum needed to reach the goal. Given a perturbation radius $\epsilon > 0$, we verify that this action is selected throughout a neighborhood of $x^{(0)}$:

$$\forall x \text{ s.t. } \|x - x^{(0)}\|_\infty \le \epsilon : \quad \big(y_{\mathcal{N}'_\delta}(x)\big)_0 \ge \big(y_{\mathcal{N}'_\delta}(x)\big)_a \quad \forall a \in \{1, 2\}.$$

We use $\epsilon = 0.05$.

(2) No left action near the goal. The goal is located on the right. In a region close to the goal in which the car already moves to the right, the controller should not select the left action, since this would move the car away from the goal. Formally,

$$\forall x \text{ s.t. } 0.45 \le x_1 \le 0.6, \;\; 0 \le x_2 \le 0.07 : \quad \max\big\{\big(y_{\mathcal{N}'_\delta}(x)\big)_1, \big(y_{\mathcal{N}'_\delta}(x)\big)_2\big\} > \big(y_{\mathcal{N}'_\delta}(x)\big)_0.$$

CartPole. We consider three properties capturing local robustness and policy correctness in critical regions of the state space. The actions are $a \in \{0, 1\}$, corresponding to pushing the cart to the left and to the right.

(1) Local action consistency. Given a nominal state $x^{(0)} \in \mathbb{R}^4$ and a relative perturbation $\epsilon > 0$, we consider a hyper-rectangle around $x^{(0)}$ whose dimension-wise bounds are proportional to the state magnitude. Let $a^\star = f_{\mathcal{N}'_\delta}(x^{(0)})$. We verify

$$\forall x \text{ s.t. } |x_i - x_i^{(0)}| \le \epsilon\,|x_i^{(0)}|, \;\; \forall i \in \{1, \ldots, 4\} : \quad \big(y_{\mathcal{N}'_\delta}(x)\big)_{a^\star} \ge \big(y_{\mathcal{N}'_\delta}(x)\big)_{1-a^\star},$$

which ensures that the selected action remains invariant under local perturbations of the input. We instantiate this property at $x^{(0)} = (-0.1, -0.1, 0.1, 0.1)$ with $\epsilon = 0.3$.

(2) Action consistency for right-leaning states. In a region where the pole leans to the right and is close to instability, the policy should push the cart to the right (action 1) to stabilize the pole. Formally,

$$\forall x \text{ s.t. } -0.2 \le x_1 \le 0.2,\; -0.02 \le x_2 \le 0.02,\; 0.18 \le x_3 \le 0.25,\; -0.02 \le x_4 \le 0.02 : \quad f_{\mathcal{N}'_\delta}(x) = 1.$$

(3) Action consistency for left-leaning states. Symmetrically, when the pole leans to the left, the policy should push the cart to the left (action 0). Formally,

$$\forall x \text{ s.t. } -0.2 \le x_1 \le 0.2,\; -0.02 \le x_2 \le 0.02,\; -0.25 \le x_3 \le -0.18,\; -0.02 \le x_4 \le 0.02 : \quad f_{\mathcal{N}'_\delta}(x) = 0.$$

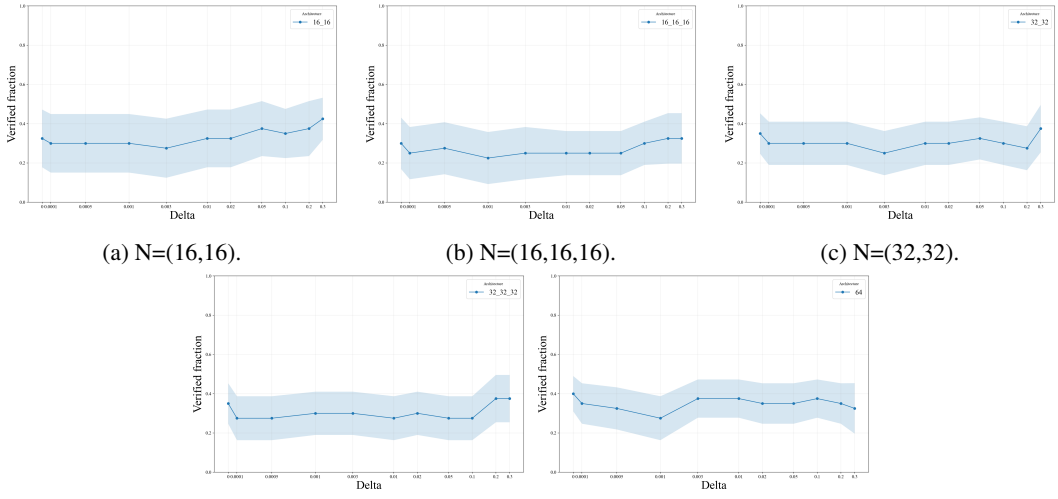
A property is counted as verified only if the verifier proves it. Properties for which the verifier returns a counterexample or no conclusive result are counted as not verified.

E TRAINING DETAILS

We train the reference controllers with DQN (Mnih et al., 2013) using the implementation of Stable-Baselines3 (Raffin et al., 2021) with its default hyperparameters, including `learning_rate` $= 10^{-4}$, `batch_size` $= 32$, and `exploration_fraction` $= 0.1$. Training runs for 2×10^7 interaction steps on MountainCar-v0, using 8 parallel environments, and for 2×10^6 interaction steps on CartPole-v1. To obtain statistically meaningful results, we consider five reference architectures per environment and train 20 independently initialized controllers for each architecture using the seeds $0, \ldots, 19$. For MountainCar-v0, the hidden-layer configurations are (64), $(16, 16)$, $(32, 32)$, $(16, 16, 16)$, and $(32, 32, 32)$. For CartPole-v1, they are (32), $(8, 8)$, $(16, 16)$, $(8, 8, 8)$, and $(16, 16, 16)$.

F ADDITIONAL EXPERIMENTAL RESULTS

This section provides the per-architecture verification results that complement the aggregate results in Section 6.1. For each environment, Figures 4 and 5 show the verified fraction as a function of the approximation parameter δ for each reference controller architecture. Each curve aggregates the independently trained controllers with the same hidden-layer configuration, and the shaded regions denote 95% confidence intervals.

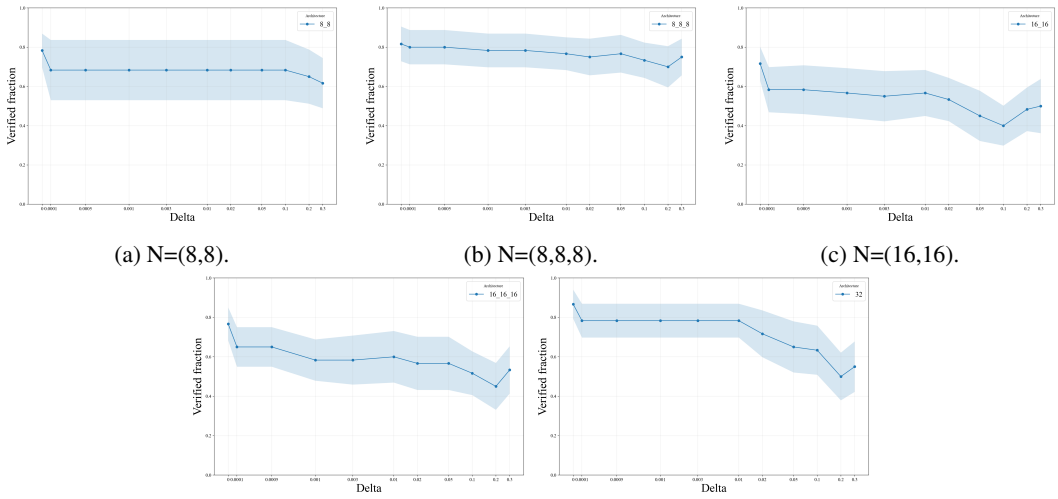


(a) N=(16,16). (b) N=(16,16,16). (c) N=(32,32).

(d) N=(32,32,32). (e) N=(64).

Figure 4: MountainCar verified fraction as a function of δ for each reference controller architecture.

The architecture-level results are consistent with the trends observed in Figure 2. For MountainCar, the verified fraction is lower overall and fluctuates between about 0.23 and 0.42 without a systematic decrease for all architectures, with an increase for some architectures at the largest tested values of δ. For CartPole, most architectures maintain a high verified fraction for small and moderate values of δ, followed by a gradual decrease as the approximation becomes coarser. These results indicate that the effect of the δ-transformation is not only task-dependent but also sensitive to the geometry of the decision trees extracted from the different reference architectures, which determines both the constant $\kappa_{\mathcal{S}}$ and the size of the disagreement region in Theorem 5.2. Nevertheless, the same qualitative conclusion holds across architectures: for moderate values of δ, the fixed-depth network $\mathcal{N}'_\delta$ obtained from the fixed tree preserves a nontrivial fraction of the verified properties, whereas larger values of δ can reduce verifiability, as observed for CartPole.



(a) N=(8,8). (b) N=(8,8,8). (c) N=(16,16).

(d) N=(16,16,16). (e) N=(32).

Figure 5: CartPole verified fraction as a function of δ for each reference controller architecture.