
VERIFYING NEURAL NETWORKS WITH REINFORCEMENT LEARNING

Hai Duong

George Mason University
Fairfax, VA, USA

Thanh Le

Unaffiliated
Yokosuka, Japan

ThanhVu Nguyen

George Mason University
Fairfax, VA, USA

ABSTRACT

Formal verification can play a key role in ensuring the reliability of Deep Neural Networks (DNNs) deployed in safety-critical systems. Modern DNN verifiers employ a branch-and-bound framework, which alternates between branching (splitting into smaller subproblems) and bounding (pruning subproblems) to efficiently explore the verification space. However, existing branching heuristics make greedy decisions based on static scoring functions. They do not anticipate long-term efficiency or leverage the growing availability of verification data to improve performance.

This work introduces RSB, a reinforcement learning framework that learns to refine baseline branching heuristics. It trains an actor-critic architecture to maximize cumulative future rewards rather than immediate scores. The actor generates attention weights from observations of raw neuron features and learned graph embeddings, which rescale baseline heuristic scores to guide neuron branching. Evaluation on 600 challenging instances demonstrates that RSB consistently outperforms state-of-the-art branching heuristics, solving 11% more instances while reducing branch exploration by 50%.

1 Introduction

Deep Neural Networks (DNNs) are increasingly being employed as components of mission-critical systems across a range of domains, *e.g.*, autonomous driving Shao et al. [2023] or medicine Morris et al. [2023]. Thus, DNNs require high levels of assurance to be deployed with confidence in such applications. To provide evidence that DNN behavior meets expectations, researchers have developed techniques for verifying that DNNs satisfy required specifications Shriver et al. [2021], Duong et al. [2026a,b]. Dozens of DNN verifiers have been reported in the literature, and a yearly competition has documented advances in their capabilities Brix et al. [2023, 2024], Kaulen et al. [2025].

As with traditional software verification, DNN verification is often framed as a satisfiability problem and also suffers from scalability challenges Katz et al. [2017], Duong et al. [2026c]. To tackle the large search space of DNN verification, modern verifiers Ferrari et al. [2022], Wang et al. [2021a], Duong et al. [2024a], Wu et al. [2024] often employ the Branch-and-Bound (BaB) Bunel et al. [2020] approach, which alternates between *bounding* (employing abstraction methods to prune branches) and *branching* (selecting neurons to split). Most advances in DNN verification focus on developing novel abstract domains to tighten the bounds computed during the bounding step Zhang et al. [2018], Wang et al. [2018], Singh et al. [2019, 2018], Tran et al. [2019], Xu et al. [2020a,b].

In contrast, the branching step—which determines the order of neuron splits and therefore significantly impacts BaB verifier efficiency—is under-explored. State-of-the-Art (SOTA) DNN verifiers often adopt branching heuristics like BABSR Bunel et al. [2020] and FSB De Palma et al. [2021] with hand-crafted scoring functions to determine the branching order. These heuristics have several major limitations: (i) they require specialized expertise to design for different problem characteristics (*e.g.*, different activation functions); (ii) they make greedy decisions at each step without anticipating long-term search efficiency; and (iii) they fail to leverage the growing availability of verification data to improve the performance.

This work introduces RSB, a Deep Reinforcement Learning (DRL) approach that addresses these limitations by learning adaptive branching policies from verification data. First, RSB automatically learns to refine any baseline branching heuristic such as FSB by training on diverse verification instances. Second, RSB formulates branching as a sequential decision-making problem Sutton et al. [1998] and trains a network Bello et al. [2016a] via DRL Mnih et al. [2016],

Haarnoja et al. [2018] to maximize cumulative future rewards (e.g., negation of numbers of branches), which enables long-term search efficiency optimization instead of greedy local decisions. Third, by leveraging available verification benchmarks Brix et al. [2024], Kaulen et al. [2025] and the ease of synthesizing additional training data, RSB follows the paradigm of modern ML where performance improves with scale as demonstrated by large language models Kaplan et al. [2020]. Finally, RSB serves as a general architecture-agnostic framework that can enhance any existing branching heuristic across diverse DNN types and activation functions by combining learned graph embeddings (to encode network structure) and raw neuron features (e.g., neuron bounds, to capture properties/specifications).

Evaluation on 600 challenging instances across networks of varying sizes and architectures shows that RSB consistently outperforms SOTA branching heuristics such as FSB by solving 11% more instances and reducing branch exploration by 50%. Furthermore, RSB generalizes beyond its training distribution, solving the most instances on unseen network architectures (e.g., CNN) and activation function (e.g., Sigmoid). We consider these improvements substantial given the maturity of existing heuristics like FSB, which have been widely adopted by leading verifiers and represent the culmination of years of research.

Our contributions are: (i) a DRL formulation that learns to refine existing baseline branching heuristic through attention weights (thereby enabling architecture-agnostic policy learning); (ii) an architecture that handles variable-sized inputs as the number of unstable neurons varies across instances and shrinks with each BaB split; (iii) a training procedure adapted to BaB’s branching dynamics that accounts for both successor subproblems per branching decision; and (iv) an implementation of RSB using an existing DNN verifier and an evaluation showing the effectiveness of the approach.

2 Preliminaries

DNN verification. Given a DNN N and a property ϕ , the *DNN verification problem* asks if ϕ is a valid property of N . Typically, ϕ is a formula of the form $\phi_{in} \Rightarrow \phi_{out}$, where ϕ_{in} is a property over the inputs of N and ϕ_{out} is a property over the outputs of N . Modern techniques often treat the DNN verification as a *satisfiability* problem Wang et al. [2021a], Ferrari et al. [2022], Duong et al. [2024b], Wu et al. [2024], Duong et al. [2024a]. Specifically, given a formula α representing N and the formula $\phi_{in} \Rightarrow \phi_{out}$ representing the desired property, a DNN verifier checks the satisfiability of the formula: $\alpha \wedge \phi_{in} \wedge \overline{\phi_{out}}$. The verifier returns *unsat* if the formula is unsatisfiable, indicating ϕ is a valid property of N , and *sat* otherwise, indicating ϕ is not valid. An *unsat* result can further be accompanied by a proof, e.g., derived from the BaB search tree, that can be independently checked Duong et al. [2026d].

Modern DNN verifiers often follow a branch-and-bound (BaB) framework Bunel et al. [2020], shown in Alg. 3 in §A). BaB iterates between two components: **DECIDE** (branching) assigns active/inactive status for a neuron and thereby splits the problem into two subproblems, and **DEDUCE** (bounding) computes the bounds of neuron outputs and checks the feasibility of the current subproblem. Both components are critical to verification efficiency, but this work focuses on improving **DECIDE**, which is responsible for branching decisions and is relatively under-explored compared to **DEDUCE** (e.g., designing new abstract domains Singh et al. [2018, 2019], Tran et al. [2019], Wang et al. [2018], Xu et al. [2020a,b], Zhang et al. [2022], Duong et al. [2024a], Ferrari et al. [2022]).

Branching (DECIDE) heuristics. At each verification step, selecting which neuron to branch on from the set of all neurons constitutes a combinatorial optimization problem. The choice of which neuron to split affects how quickly the search space is explored and pruned, e.g., determining the numbers of steps and the verification time. A well-designed decision heuristic can dramatically reduce the search space, while a poor one may lead to exponential growth in numbers of subproblems.

SOTA heuristic FSB De Palma et al. [2021] estimates improvement for each neuron if it is branched on, and is widely used by leading DNN verifiers. However, FSB only considers local optimality at each step, not long-term search efficiency, and relies on domain expert knowledge, e.g., Lagrangian computations. In contrast, RSB learns to generate weights that modify baseline heuristic’s scores (e.g., FSB) to optimize long-term verification efficiency, thus, produces more effective decisions overall (see §6).

Deep Reinforcement Learning. DRL combines Reinforcement Learning (RL) with DNN to learn sequential decision-making policies from interaction with an environment. An RL problem is formalized as a Markov Decision Process $\langle S, A, P, R \rangle$, where S is the state space, A the action space, P the transition function, and $R(s, a)$ the reward function. A DRL agent learns a policy to maximize the expected cumulative reward over time. This work uses an actor-critic algorithm Haarnoja et al. [2018], which maintains two networks: an actor that selects actions and a critic that estimates their value. The actor is updated to take actions with higher estimated value, while the critic is updated to better predict future rewards. We refer to §B for a detailed actor and critic objectives.

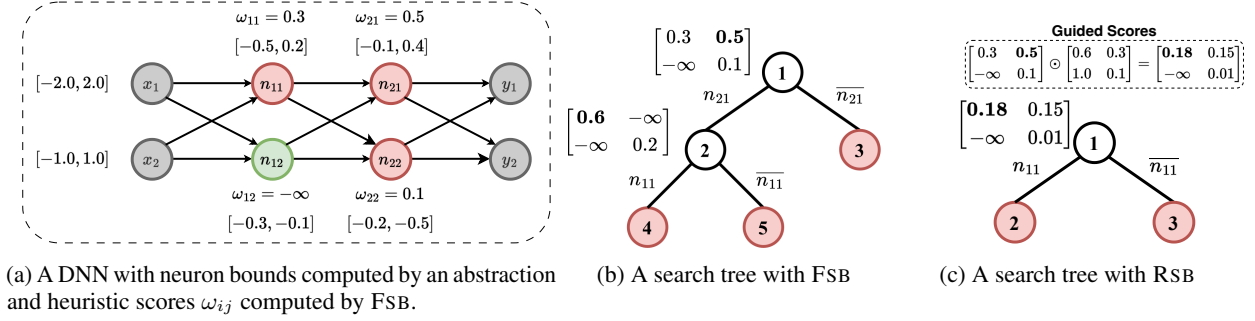


Fig. 1: Comparison of different search trees performed by FSB and RSB for verifying the property in Eq. 1.

3 Motivating example

We illustrate RSB using a concrete example. Consider the DNN N shown in Fig. 1a, which consists of two linear layers with ReLU activations. We aim to verify the property:

$$\phi \equiv (-2 \leq x_1 \leq 2 \wedge -1 \leq x_2 \leq 1) \Rightarrow (y_1 > y_2) \quad (1)$$

BAB_{NNV} computes a bound estimation of hidden neurons (e.g., using an abstraction as LiRPA Xu et al. [2020a]) to determine neuron stability. This step reveals three *unstable* ReLU neurons (shown in red in Fig. 1a): n_{11} , n_{21} , and n_{22} . For example, n_{11} is unstable because its bounds $[-0.5, 0.2]$ mean it could be either active or inactive depending on the input. BAB_{NNV} splits (branches) each unstable neuron into positive (active) and negative (inactive) branches.

SOTA BAB_{NNV} employs various branching heuristics to guide the search, e.g., FSB De Palma et al. [2021]. A branching heuristic assigns scores ω to unstable neurons to prioritize which neuron to split next. Fig. 1b shows a search tree of verifying the property in Eq. 1 using BAB_{NNV} with FSB, a popular heuristic used by major DNN verifiers including $\alpha\beta$ -CROWN Wang et al. [2021a], GCP-CROWN Zhang et al. [2022] and NEURALSAT Duong et al. [2024a].

First, BAB_{NNV} computes FSB scores for unstable neurons as $\omega_{11} = 0.3$, $\omega_{21} = 0.5$, and $\omega_{22} = 0.1$ and selects n_{21} for the initial branching because it has the highest score. After this branching, it proves unsatisfiability of the negative branch $\bar{n}_{21}$ while the positive branch is still undecided. Next, BAB_{NNV} computes FSB scores for the remaining two unstable neurons and obtains $\omega_{11} = 0.6$ and $\omega_{22} = 0.2$, and selects n_{11} for the second branching decision based on its score. BAB_{NNV} then can prove both branches and completes the verification. In total, BAB_{NNV} with FSB requires 2 steps and processes 4 branches to verify the property.

RSB improves this by learning to generate new weights to adjust the original FSB scores. Neuron-level features are constructed from four available *raw features* of unstable neurons n_{11} , n_{21} , n_{22} : lower bound, upper bound, bias (0.0 for simplicity), and mask (1 for unstable, 0 for stable):

$$\mathbf{x} = \text{GETOBSERVATION} \left(\begin{bmatrix} -0.5 \\ 0.2 \\ 0.0 \\ 1.0 \end{bmatrix}, \begin{bmatrix} -0.1 \\ 0.4 \\ 0.0 \\ 1.0 \end{bmatrix}, \begin{bmatrix} -0.2 \\ 0.5 \\ 0.0 \\ 1.0 \end{bmatrix} \right) \quad (2)$$

GETOBSERVATION combines neuron features with Graph Neural Networks (GNN) embeddings to produce observation $\mathbf{x}$ (see §5.1). Next, the actor π_ψ takes as input $\mathbf{x}$ and generates weights:

$$\begin{bmatrix} a_{n_{11}} \\ a_{n_{21}} \\ a_{n_{22}} \end{bmatrix} = \pi_\psi(\mathbf{x}) = \begin{bmatrix} 0.6 \\ 0.3 \\ 0.1 \end{bmatrix} \quad (3)$$

The precomputed FSB scores ω_{ij} are then scaled by the weights a_{ij} to produce the guided scores:

$$\begin{bmatrix} \omega'_{11} \\ \omega'_{21} \\ \omega'_{22} \end{bmatrix} = \begin{bmatrix} a_{n_{11}} \\ a_{n_{21}} \\ a_{n_{22}} \end{bmatrix} \odot \begin{bmatrix} \omega_{11} \\ \omega_{21} \\ \omega_{22} \end{bmatrix} = \begin{bmatrix} 0.6 \\ 0.3 \\ 0.1 \end{bmatrix} \odot \begin{bmatrix} 0.3 \\ 0.5 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 0.18 \\ 0.15 \\ 0.01 \end{bmatrix} \quad (4)$$

This reweighting flips the priority, e.g., n_{11} now has the highest score ($\omega'_{11} = 0.18$) instead of n_{21} ($\omega'_{21} = 0.15$). As shown in Fig. 1c, branching on n_{11} first immediately proves both branches (positive n_{11} and negative $\bar{n}_{11}$ are UNSAT), completing the verification in just one step (2 branches) instead of two steps (4 branches) when using FSB.

Key intuition. By learning from existing DNN verification tasks, RSB determines that certain neurons tend to be more effective branching points than others. For example, branching on n_{11} is preferred over n_{21} because n_{11} lies in an earlier layer and has a wider pre-activation interval, causing its split to simultaneously stabilize multiple downstream neurons. In particular, branching on n_{11} constrains both n_{21} and n_{22} in both subproblems, leading to tighter intermediate bounds that allow verification to terminate quicker. In contrast, n_{21} (chosen by FSB) is only weakly coupled to n_{22} ; branching on it leaves n_{22} unstable, requiring additional splits. Through training on many networks and properties, RSB learns this recurring pattern and selects neurons that are more strongly connected, even if their immediate heuristic scores (e.g., computed by FSB) are not maximal.

4 Branching as DRL environment

We map the branching problem in BAB_{NNV} to a DRL environment Sutton et al. [1998], defining a state space $\mathcal{S}$, observation space $\mathcal{O}$, action space $\mathcal{A}$, and reward function R .

State space. Each state $\mathbf{s} \in \mathcal{S}$ comprises the verification instance $\langle N, \phi_{in}, \phi_{out} \rangle$ and the current assignment σ encoding accumulated branching decisions. When BAB_{NNV} branches on neuron n_{ij} , it generates two successor states:

$$\mathbf{s}' = \langle N, \phi_{in}, \phi_{out}, \sigma \wedge n_{ij} \rangle \quad (\text{positive branch}) \quad (5)$$

$$\bar{\mathbf{s}}' = \langle N, \phi_{in}, \phi_{out}, \sigma \wedge \bar{n}_{ij} \rangle \quad (\text{negative branch}) \quad (6)$$

where n_{ij} and $\bar{n}_{ij}$ denote the neuron being active and inactive, respectively. This dual-branch structure means each branching decision expands the search tree exponentially, which distinguishes DNN verification from single-successor combinatorial optimization problems.

Observation space. The full state $\mathbf{s}$ cannot be directly processed by the DRL agent due to its complex, variable-sized structure. We therefore define an observation space $\mathcal{O}$ and an extraction function GETOBSERVATION that maps $\mathbf{s}$ to a compact, agent-compatible representation satisfying three criteria: (1) *generalizability* across DNN architectures, (2) *expressivity* to capture neuron interdependencies, and (3) *scalability* by attending only to unstable neurons.

Formally, $\mathcal{O} \equiv \mathbb{R}^{|\mathcal{N}_u| \times (F+E)}$, where $\mathcal{N}_u$ is the set of unstable neurons, and F, E are the raw feature and embedding dimensions. For each unstable neuron n_{ij} , raw features include the activation bounds (l_{ij}, u_{ij}) , bias b_{ij} , and a binary mask m_{ij} indicating whether the neuron has been branched on previously. These local features are augmented with learned embeddings from a GNN that encodes the global structure of the verification problem (§5.1).

Action space. Existing branching heuristics such as BABSR and FSB rank unstable neurons by computing a score ω_{ij} for each and greedily selecting $n^* = \arg\max_{i,j} \omega_{ij}$, yielding locally optimal decisions. RSB improves on this by learning to adjust these scores rather than replacing them. We define the action space as $\mathcal{A} \equiv [0, 1]^{|\mathcal{N}_u|}$, where each $a_{ij} \in [0, 1]$ is a learned weight for neuron n_{ij} . Given observation $\mathbf{o}$, the actor samples $\mathbf{a} \sim \pi_\psi(\mathbf{a}|\mathbf{o})$ and selects:

$$n^* = \arg\max_{i,j} \omega_{ij} \odot a_{ij} \quad (7)$$

This allows the agent to refine any heuristic’s priorities based on the current state and promote long-term branching efficiency.

Reward function. The reward function $R(\mathbf{s}, \mathbf{a})$ incentivizes efficient verification by penalizing unverified subproblems. Formally, the agent incurs a unit reward for each resolved subproblem:

$$R(\mathbf{s}, \mathbf{a}) = \sum_{\sigma' \in \{\sigma \wedge n^*, \sigma \wedge \bar{n}^*\}} \neg \text{DEDUCE}(N, \phi_{in}, \phi_{out}, \sigma') \quad (8)$$

where $\langle N, \phi_{in}, \phi_{out}, \sigma \rangle \leftarrow \mathbf{s}$, and n^* is selected neuron obtained from $\mathbf{a}$ (Eq. 7). Since each verified subproblem contributes a +1 reward, maximizing the cumulative discounted reward is equivalent to minimizing the total number of subproblems generated throughout the verification process. The overall goal is to learn a policy $\pi_\psi(\mathbf{a}|\mathbf{o})$ that maximizes the expected cumulative discounted reward $\mathbb{E}_{\pi_\psi} \left[\sum_{t=0}^T \gamma^t R(\mathbf{s}_t, \mathbf{a}_t) \right]$, where γ is the discount factor.

Theorem 1 (Sound and Complete). RSB *preserves* BAB_{NNV} ’s *soundness and completeness*.

Proof. RSB only affects the neuron selection order via Eq. 7, leaving the underlying verification logic of BAB_{NNV} intact. Since BAB_{NNV} exhaustively explores all necessary branches regardless of branching order, the verification result is unchanged. $\square$

Alg. 1: BAB_{NNV} with RSB

input : DNN N , property ϕ , trained actor π_ψ
output : unsat if property is valid, otherwise sat.

```

1  $queue \leftarrow \{\emptyset\}$ 
2 while  $queue$  do
3    $\sigma \leftarrow \text{SELECT}(queue)$ 
4   if  $\text{DEDUCE}(N, \phi, \sigma)$  then
5      $\mathbf{o}_t \leftarrow \text{GETOBSERVATION}(N, \phi, \sigma)$ 
6      $\mathbf{a}_t \sim \pi_\psi(\mathbf{a}|\mathbf{o}_t)$ 
7      $\omega_t \leftarrow \text{GETBRANCHSCORE}(N, \phi, \sigma)$ 
8      $n^* \leftarrow \arg \max_{ij} \mathbf{a}_t \odot \omega_t$ 
9      $queue \leftarrow queue \cup \{\sigma \wedge n^* ; \sigma \wedge \overline{n^*}\}$ 
10 return unsat

```

Alg. 2: Training RSB

input : Training set $\{(N_i, \phi_i)\}$
output : Trained actor π_ψ , critics Q_{ϕ_1}, Q_{ϕ_2}

```

1  $\mathcal{D} \leftarrow \emptyset$ 
2 for  $e$  in  $[1, \dots, \text{NumEpisodes}]$  do
3    $(N, \phi) \leftarrow \text{SAMPLEINSTANCE}(\{(N_i, \phi_i)\})$ 
4    $\mathbf{o}_0 \leftarrow \text{GETOBSERVATION}(N, \phi)$ 
5   for  $t$  in  $[0, \dots, T]$  do
6      $\omega_t \leftarrow \text{GETBRANCHSCORE}(N, \phi_i, \sigma)$ 
7      $\mathbf{a}_t \sim \pi_\psi(\mathbf{a}|\mathbf{o}_t)$ 
8      $n^* \leftarrow \arg \max_{ij} \mathbf{a}_t \odot \omega_t$ 
9      $\mathbf{o}'_{t+1}, \bar{\mathbf{o}}'_{t+1} \leftarrow \text{DECIDE}(n^*)$ 
10     $m_{t+1}, \bar{m}_{t+1} \leftarrow \text{DEDUCE}(\mathbf{o}'_{t+1}, \bar{\mathbf{o}}'_{t+1})$ 
11     $r_t \leftarrow -|\{\mathbf{o}' : m' = 1\}|$ 
12     $\mathcal{D}.add(\{(\mathbf{o}_t, \mathbf{a}_t, r_t, \mathbf{o}'_{t+1}, \bar{\mathbf{o}}'_{t+1}, m_{t+1}, \bar{m}_{t+1})\})$ 
13     $\pi_\psi, Q_{\phi_1}, Q_{\phi_2} \leftarrow \text{OPTIMIZE}(\mathcal{D})$ 
14 return  $\pi_\psi, Q_{\phi_1}, Q_{\phi_2}$ 

```

5 The RSB approach

Alg. 1 shows how RSB integrates into BAB_{NNV}. At each branching step, GETOBSERVATION extracts the current observation $\mathbf{o}_t$ (line 5), the actor π_ψ generates attention weights $\mathbf{a}_t$ (line 6), and GETBRANCHSCORE computes baseline scores ω_t for unstable neurons (line 7). The neuron maximizing the guided score $\arg \max_{ij} \mathbf{a}_t \odot \omega_t$ is selected for branching (line 8); the rest of the procedure is unchanged from BAB_{NNV}.

RSB primarily relies on three components: a GNN that enriches raw neuron features with structural embeddings (§5.1), an actor-critic architecture that learns to refine branching scores (§5.2), and an off-policy training procedure with modifications for the dual-branch structure of BAB_{NNV} (§5.3).

5.1 Observation construction

We train a Graph Convolution Networks (GCN) Kipf and Welling [2017] to enrich raw neuron features with learned embeddings that capture network-wide structural patterns. The graph is constructed with all neurons as nodes, activation bounds, bias, and branching status ($l_{ij}, u_{ij}, b_{ij}, m_{ij}$) as node features, network connections as edges forming adjacency matrix A , and heuristic scores (e.g., FSB) as supervision labels for unstable neurons. This yields a node regression problem where the GCN predicts branching scores for unstable neurons $\mathcal{N}_u$, producing embeddings $\mathbf{o}_{\text{embed}} \in \mathbb{R}^{|\mathcal{N}_u| \times E}$.

Since the raw feature dimension $F \ll E$, directly concatenating raw features with embeddings would bias representations toward global information. We therefore project raw features $\mathbf{o}_{\text{raw}} \in \mathbb{R}^{|\mathcal{N}_u| \times F}$ to dimension E , concatenate with $\mathbf{o}_{\text{embed}}$, and project again to produce the unified observation $\mathbf{x} \in \mathbb{R}^{|\mathcal{N}_u| \times E}$ that balances local and global information.

5.2 Actor-Critic architecture

We employ an actor-critic Haarnoja et al. [2018] framework for its sample efficiency Wen et al. [2021], Tan et al. [2025], which is important since each verification episode is computationally expensive Katz et al. [2017]. Both actor $\pi_\psi(\mathbf{a}|\mathbf{o})$ and critics $Q_\phi(\mathbf{o}, \mathbf{a})$ use PointerNet Bello et al. [2016a] architectures that naturally handle variable-length inputs (the number of unstable neurons changes across instances), taking the unified observation $\mathbf{x}$ from §5.1 as input.

The actor (Fig. 2a) uses an encoder-decoder with an attention-based pointer mechanism. An encoder processes $\mathbf{x}$ to produce reference vectors $\mathbf{r} \in \mathbb{R}^{|\mathcal{N}_u| \times H}$ capturing relationships among all neurons. The decoder, initialized with a learnable state $\mathbf{x}_0^{\text{dec}} \in \mathbb{R}^E$, generates query $\mathbf{q} \in \mathbb{R}^H$, which PointerNet uses to compute attention scores output as $\mathbf{a} \in \mathbb{R}^{|\mathcal{N}_u|}$. The feature of the selected neuron becomes the next decoder state, enabling decisions that condition on prior selections.

The critic (Fig. 2b) mirrors the actor’s encoder but differs in decoder initialization. Instead of a learnable state, the decoder is initialized with the actor’s decoder state $\mathbf{x}_t^{\text{dec}}$ from the current branching step. This produces Q-values $Q(\mathbf{o}, \mathbf{a}) \in \mathbb{R}^{|\mathcal{N}_u|}$ for all neurons simultaneously. Twin critics Q_{ϕ_1}, Q_{ϕ_2} share this architecture with independent parameters to reduce variance in value estimation.

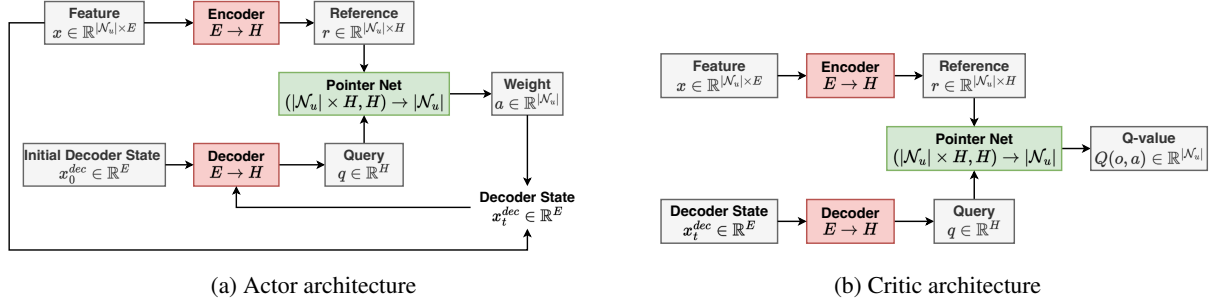


Fig. 2: The actor and critic architectures in RSB.

5.3 Training procedure

Alg. 2 presents the training procedure. Some functions of BAB_{NNV} remain (e.g., SELECT), thus, are hidden for simplicity. For each episode, a verification instance (N, ϕ) is sampled from the training set (line 3) and the initial observation $\mathbf{o}_0$ is extracted (line 4). At each branching step, the actor samples attention weights $\mathbf{a}_t \sim \pi_\psi(\mathbf{a}|\mathbf{o}_t)$ (line 7), which are combined with heuristic scores ω_t to select the branching neuron n^* (line 8). DECIDE branches on n^* and returns two successor observations $\mathbf{o}'_{t+1}, \bar{\mathbf{o}}'_{t+1}$. Next, DEDUCE runs abstraction on each branch and returns termination masks $m_{t+1}, \bar{m}_{t+1}$ indicating which branches are solved. The reward r_t penalizes unsolved branches (line 11), and a transition is stored in the replay buffer $\mathcal{D}$ (line 12). OPTIMIZE then samples a batch from $\mathcal{D}$ and updates the actor and critic networks (line 13). Two adaptations are needed to handle the dual-branch structure of BAB_{NNV} .

Replay buffer. We extend the standard transition Haarnoja et al. [2018] to capture both successor subproblems:

$$\mathcal{D} = \{\dots, \langle \mathbf{o}_t, \mathbf{a}_t, r_t, \mathbf{o}'_{t+1}, \bar{\mathbf{o}}'_{t+1}, m_{t+1}, \bar{m}_{t+1} \rangle, \dots\} \quad (9)$$

where $\mathbf{o}'_{t+1}, \bar{\mathbf{o}}'_{t+1}$ are the positive and negative branch observations, and $m_{t+1}, \bar{m}_{t+1} \in \{0, 1\}$ are termination masks indicating whether each branch is immediately resolved by DEDUCE .

Loss function. Since each branching action produces two successor subproblems, the critic loss adapts the Bellman equation to incorporate both branches weighted by their termination masks:

$$\mathcal{L}_Q(\phi) = \mathbb{E}_{\tau_t \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\phi(\mathbf{o}_t, \mathbf{a}_t) - \left(r_t + \gamma \left(m_{t+1} \mathbb{E}_{\mathbf{a} \sim \pi_\psi(\mathbf{a}|\mathbf{o}_{t+1})} [Q_{\tilde{\phi}}(\mathbf{o}_{t+1}, \mathbf{a})] + \bar{m}_{t+1} \mathbb{E}_{\mathbf{a} \sim \pi_\psi(\mathbf{a}|\bar{\mathbf{o}}_{t+1})} [Q_{\tilde{\phi}}(\bar{\mathbf{o}}_{t+1}, \mathbf{a})] \right) \right)^2 \right] \quad (10)$$

where $m_{t+1} = 0$ means the branch is solved by bounding step and contributes no future value. The target network $Q_{\tilde{\phi}}$ is updated via exponential moving average to stabilize temporal difference target. The actor is updated using policy gradients to maximize expected Q -values:

$$\mathcal{L}_\pi(\psi) = \mathbb{E}_{\mathbf{o}_t \sim \mathcal{D}} [\mathbb{E}_{\mathbf{a} \sim \pi_\psi(\mathbf{a}|\mathbf{o}_t)} [-Q_\phi(\mathbf{o}_t, \mathbf{a})]] \quad (11)$$

Training alternates between updating the critic to improve value estimates and updating the actor to select actions with higher estimated returns.

6 Evaluation

6.1 Experimental design

Dataset and training. To train RSB, we generated 480,960 FNN-based verification instances with varying layers and perturbation radii. To select challenging instances that require non-trivial branching¹, we applied a two-stage filter using: (i) adversarial attack PGD Madry et al. [2017] to remove SAT instances, and (ii) abstraction LIRPA Xu et al. [2020a] to remove easy UNSAT instances. This yielded 14,454 challenging instances used for training. We train RSB for 100,000 episodes with batch size 512 and gradient accumulation over 8 steps (Tab. 3) and a 2-layer GCN with 128 hidden dimensions for embeddings.

¹In fact, out of 2700 instances in regular track of VNN-COMP'25 Kaulen et al. [2025], 92% problems (2473/2700) were solved without branching or very few branching steps.

Tab. 1: Verification instances.

Benchmark	Layers	Types	Neurons	Parameters	Instances
FNN	4-6	FC, ReLU	0.5K-1.5K	269K-532K	200
CNN	4-6	Conv, FC, ReLU	3.2K-6.8K	55K-215K	200
Sigmoid	5	FC, ReLU, Sigmoid	0.3K	343K	200
Total					600

Tab. 2: Results across benchmarks (instances solved by at least one method).

Method	FNN			CNN			Sigmoid		
	Time (s)	Branch ($\times 10^6$)	Solved (#)	Time (s)	Branch ($\times 10^6$)	Solved (#)	Time (s)	Branch ($\times 10^6$)	Solved (#)
POLARITY	6064.32	170.81	1	8662.55	583.01	0	6027.70	62.98	10
UPB	4359.36	27.17	23	2387.48	1.73	71	3963.69	8.59	39
FSB	2679.00	6.65	43	2295.89	1.81	71	4336.73	8.99	34
RSB	2051.94	1.86	50	2419.58	1.69	72	3450.80	5.14	43

Verification problems. We use 600 challenging verification problems across three network architectures (Tab. 1). In addition to 200 FNN-based instances, we evaluate on 200 CNN-based De Palma et al. [2021] and 200 Sigmoid-based Kaulen et al. [2025] instances, whose networks are not used in training, for generalization.

Baselines. We compare RSB against (i) FSB De Palma et al. [2021], which estimates bound improvements via Lagrangian Decomposition and is used in $\alpha\beta$ -CROWN Wang et al. [2021a], Zhang et al. [2022] (ranked 1st in VNN-COMPs—the annual DNN verification competition to compare different approaches and showcase the latest advances in the field Bak et al. [2021], Brix et al. [2023], Kaulen et al. [2025], Brix et al. [2024]); and NEURALSAT Duong et al. [2024a, 2025], Duong and Nguyen [2025] (2nd in VNN-COMP’24 and ’25); (ii) UPB De Palma et al. [2022], a fast approximation of FSB; and (iii) POLARITY Wu et al. [2020], which uses bound differences directly and is used in MARABOU Wu et al. [2024] (ranked 2nd in VNN-COMP’23). All methods are integrated into NEURALSAT for a fair comparison.

Experimental setup. Our experiments were conducted on a Linux machine with an AMD Ryzen Threadripper 32-Core, 128GB RAM, and an NVIDIA GeForce RTX 4090, 24GB VRAM. We use timeout 120s per instance, which follows the rules in recent VNN-COMPs Brix et al. [2024], Kaulen et al. [2025] (6h per benchmark).

6.2 RSB performance compared to baselines

Tab. 2 presents our results. RSB solves the most instances across all three benchmarks (165 total vs. 148 for FSB, 110 for UPB, and 11 for POLARITY), an overall improvement of 11% over FSB, while exploring 50% fewer branches compared to FSB. These 11% gain in solved instances and reduction in branch exploration show that RSB allows far fewer splits by selecting “right” neurons that tighten bounds faster and prune infeasible subproblems earlier.

On FNN, RSB verifies 50 instances compared to 43 for FSB (16% more) while reducing total branch exploration by 72% (1.86M vs. 6.65M). On Sigmoid, the gains are even larger: 43 vs. 34 solved instances (26% more) with 43% fewer branches (5.14M vs. 8.99M). On CNN, the improvement is smaller (72 vs. 71 solved, 7% fewer branches), as CNNs have more neurons (Tab. 1) and both methods either timeout or require many branches to verify. Still, RSB reduces branches and solves one more instance. These results show that RSB adjusts FSB’s decisions to explore better paths.

POLARITY solves only 11 instances in total despite exploring far more branches than FSB and RSB. This shows that exploring more branches does not lead to more solved instances and that the branching heuristic has a large impact on verification performance.

We note that the chosen baselines are already strong heuristics, *e.g.*, FSB being used in leading verifiers such as $\alpha\beta$ -CROWN Wang et al. [2021a], GCP-CROWN Zhang et al. [2022], and NEURALSAT Duong et al. [2024a]. We consider improving over such well-established baselines is significantly more challenging, and these gains are meaningful progress in pushing the boundaries of DNN verification. More importantly, as a learning-based method, RSB continues to improve with more training data, better DRL algorithms, and new network architectures, unlike fixed hand-crafted heuristics.

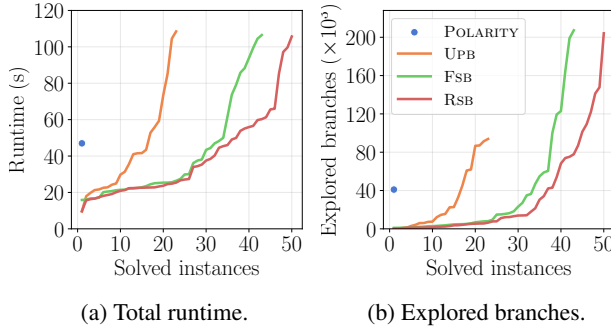


Fig. 3: Performances on seen networks.

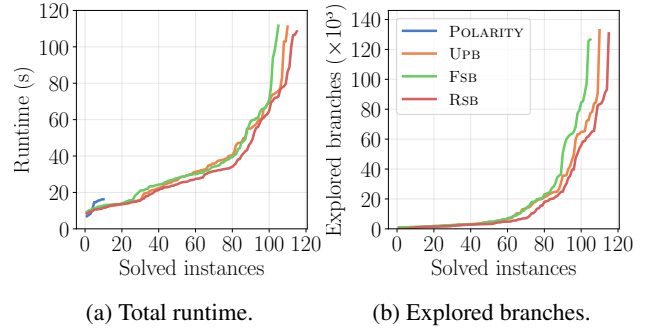


Fig. 4: Performances on unseen networks.

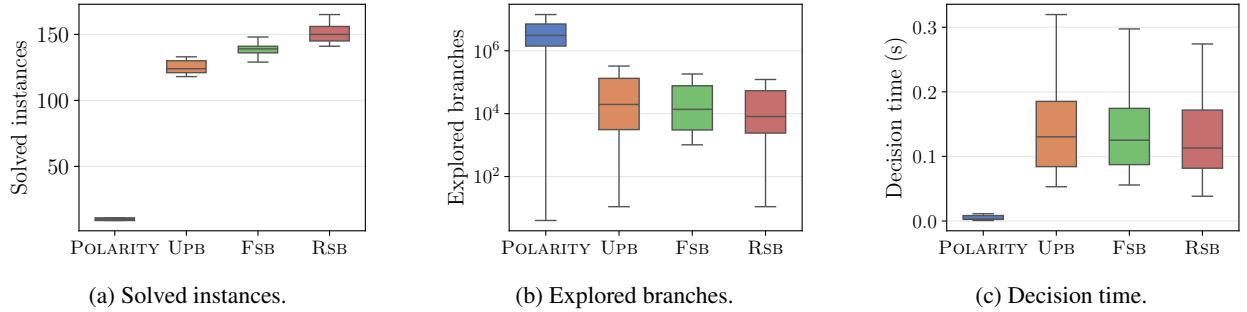


Fig. 5: Statistical distributions of computational metrics.

6.3 RSB’s generalizability

Fig. 3 illustrates the performance of RSB on seen networks (FNN). Evaluation instances share the same architectures as training instances but are paired with different properties. Since a *verification problem* is a (network, property) pair, different properties yield different hidden-layer bounds and different BaB trees—they are distinct problems—and thus train and test sets do not overlap.

RSB’s line (red) extends furthest on the x-axis (50 instances) and runs below FSB throughout both runtimes (Fig. 3a) and explored branches (Fig. 3b), which shows that RSB solves more instances while consistently requiring fewer branches or runtimes. As problem difficulty increases, the gap between RSB and FSB widens. RSB’s curve stays lower while FSB’s rises faster, indicating that the learned policy becomes more effective on hard instances where branching order has a larger impact.

Fig. 4 shows the performances on unseen architectures (CNN and Sigmoid). RSB’s line extends furthest (115 instances), with UPB second (110) and FSB third (105). An interesting observation is that FSB performs worse than UPB on unseen architectures. However, RSB adjusts FSB’s scores through learned weights, and its curve remains consistently below UPB’s across the entire range, recovering from FSB’s weakness and surpassing all baselines. This shows that RSB generalizes beyond its training distribution, learning branching strategies that transfer to unseen network architectures.

6.4 RSB’s efficiency

To assess statistical significance (Fig. 5), we evaluate each method on five runs that use different seeds for benchmark generation. RSB solves the most instances across all runs (Fig. 5a), with a median of 150 and a tight interquartile range, compared to 140 for FSB and 125 for UPB. The narrow spread of RSB’s box confirms that its advantage is consistent and not driven by a single run.

We further show the branch distribution in log scale in Fig. 5b. POLARITY explores orders of magnitude more branches ($\sim 10^6$) than the rest, yet solves the fewest instances, which clearly indicates that exploring more branches does not compensate for poor branching decisions. RSB and FSB both have medians near 10^4 , but RSB’s box is lower and its spread is tighter, showing that it reaches solutions with fewer branches across instances of varying difficulty.

Fig. 5c shows that RSB’s per-step decision time ($\sim 0.11s$) is lower than FSB and UPB ($\sim 0.13s$) despite performing additional DRL inference at every step. Both RSB and FSB run FSB scoring internally, but RSB’s better branching keeps the subproblem queue smaller (e.g., 32 vs. 64 active subproblems per batch). As a result, RSB runs FSB on fewer subproblems per step, and the time saved from the smaller batch offsets the added DRL inference cost.

7 Related work

Abstraction in BAB_{NNV} plays a key role in scalability, e.g., intervals Wang et al. [2018], zonotopes Singh et al. [2018], polytopes Singh et al. [2019], Zhang et al. [2018], Xu et al. [2020a], and star sets Tran et al. [2019] are widely used by leading verifiers Brix et al. [2023, 2024], Kaulen et al. [2025]. Bound tightening is further strengthened by multi-neuron relaxations Ferrari et al. [2022], cutting planes Zhang et al. [2022], Zhou et al. [2024], and neuron stability Duong et al. [2024a], while training for verification reduces the number of unstable neurons in the first place Xu et al. [2024]. RSB focuses on the branching decision and complements these approaches.

FSB De Palma et al. [2021] is the SOTA heuristic that estimates bound improvements via Lagrangian Decomposition, outperforming BABSR, POLARITY, and GNN heuristic Lu and Kumar [2020]. FSB estimates per-neuron branching scores using tighter bounds from the dual problem, making it a strong greedy heuristic. UPB De Palma et al. [2022] is an approximation of FSB De Palma et al. [2021] by reusing dual variables computed during the bounding steps Wang et al. [2021a]. BABSR Bunel et al. [2020] uses an abstraction to approximate bound improvements per neuron while POLARITY Wu et al. [2020] selects neurons based solely on their bounds. GNN heuristic Lu and Kumar [2020] takes a learning-based approach with bidirectional updates to predict decisions, and is closely related to our work. However, these heuristics make greedy decisions, scoring each neuron based on immediate improvements without considering long-term impact on search tree. A neuron that maximizes immediate bounds may still produce subproblems that require more branches to resolve, resulting in deep search trees.

In SAT solving, prior DRL-based approaches Kurin et al. [2020], Wang et al. [2021b], Zhai and Ge [2025] require a separate GNN forward pass at each decision step, incurring significant overhead. The one-shot guidance approach Tönshoff and Grohe [2025] reduces this cost by applying a single GNN forward pass whose predictions guide all subsequent steps, achieving a $2\times$ speedup and generalizing to larger problems.

Our work addresses the greedy selections Bunel et al. [2020], De Palma et al. [2021] by generating weights to refine baseline heuristic scores with learned policies. Unlike these methods, we use raw neuron features for architecture-agnostic verification (§5.1). We also adapt the training procedure to dual-branch dynamics, which complements the one-shot guidance approach Tönshoff and Grohe [2025] (§5.3).

8 Conclusion

We demonstrated that the proposed DRL branching heuristics can effectively learn policies that outperform mature heuristics in DNN verification. Experimental results validate that DRL can capture long-term search strategies and outperform greedy heuristics. More broadly, learning-based paradigm opens new directions for improving verification efficiency. As DRL algorithms advance Schulman et al. [2017], Haarnoja et al. [2018], Levine et al. [2020], Le et al. [2024] and training data grows, performance can continue to improve. Future work includes extending RSB to input-space branching Wang et al. [2018] for low-dimensional inputs and online adaptation during verification Bello et al. [2016b] to enhance the performance of the learned policy on hard instances.

References

- Hao Shao, Letian Wang, Ruobing Chen, Hongsheng Li, and Yu Liu. Safety-enhanced autonomous driving using interpretable sensor fusion transformer. In *Conference on Robot Learning*, pages 726–737. PMLR, 2023.
- Miranda X Morris, Aashish Rajesh, Malke Asaad, Abbas Hassan, Rakan Saadoun, and Charles E Butler. Deep learning applications in surgery: Current uses and future directions. *The American Surgeon*, 89(1):36–42, 2023. doi:10.1177/00031348221101490.
- David Shriver, Sebastian Elbaum, and Matthew B Dwyer. Reducing DNN Properties to Enable Falsification with Adversarial Attacks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 275–287. IEEE, 2021. doi:10.1109/ICSE43902.2021.00036.
- Hai Duong, Thanh Le, Lam Nguyen, and ThanhVu Nguyen. Verifying structural robustness of deep neural network. *Proceedings of the ACM on Software Engineering*, 3(FSE):1492–1514, 2026a.

- Hai Duong, Lam Nguyen, Thanh Le, and ThanhVu Nguyen. Verifying neural network robustness with dual perturbations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 27916–27925, 2026b.
- Christopher Brix, Mark Niklas Müller, Stanley Bak, Taylor T Johnson, and Changliu Liu. First three years of the International Verification of Neural Networks Competition (VNN-COMP). *International Journal on Software Tools for Technology Transfer*, pages 1–11, 2023. doi:<https://doi.org/10.48550/arXiv.2301.05815>.
- Christopher Brix, Stanley Bak, Taylor T Johnson, and Haoze Wu. The Fifth International Verification of Neural Networks Competition (VNN-COMP 2024): Summary and results. *arXiv preprint arXiv:2412.19985*, 2024.
- Konstantin Kaulen, Tobias Ladner, Stanley Bak, Christopher Brix, Hai Duong, Thomas Flinkow, Taylor T Johnson, Lukas Koller, Edoardo Manino, ThanhVu H Nguyen, et al. The 6th international verification of neural networks competition (vnn-comp 2025): Summary and results. *arXiv preprint arXiv:2512.19007*, 2025.
- Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017. doi:10.1007/978-3-319-63387-9_5.
- Hai Duong, David Shriver, ThanhVu Nguyen, and Matthew Dwyer. Compositional neural network verification via assume-guarantee reasoning. *Advances in Neural Information Processing Systems*, 38:64158–64182, 2026c.
- Claudio Ferrari, Mark Niklas Mueller, Nikola Jovanović, and Martin Vechev. Complete Verification via Multi-Neuron Relaxation Guided Branch-and-Bound. In *International Conference on Learning Representations*, 2022. doi:10.48550/arXiv.2205.00263.
- 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 Complete and Incomplete Neural Network Robustness Verification. *Advances in Neural Information Processing Systems*, 34:29909–29921, 2021a. doi:10.48550/arXiv.2103.06624.
- Hai Duong, Dong Xu, ThanhVu Nguyen, and Matthew B Dwyer. Harnessing neuron stability to improve dnn verification. *Proceedings of the ACM on Software Engineering*, 1(FSE):859–881, 2024a.
- Haoze Wu, Omri Isac, Aleksandar Zeljić, Teruhiro Tagomori, Matthew Daggitt, Wen Kokke, Idan Refaeli, Guy Amir, Kyle Julian, Shahaf Bassan, et al. Marabou 2.0: a versatile formal analyzer of neural networks. In *International Conference on Computer Aided Verification*, pages 249–264. Springer, 2024.
- Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and M Pawan Kumar. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(42):1–39, 2020.
- Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *Advances in Neural Information Processing Systems*, 31, 2018. doi:<https://dl.acm.org/doi/10.5555/3327345.3327402>.
- Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1599–1614, 2018. doi:<https://dl.acm.org/doi/10.5555/3277203.3277323>.
- Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, 2019. doi:10.1145/3290354.
- Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. *Advances in Neural Information Processing Systems*, 31, 2018. doi:<https://dl.acm.org/doi/10.5555/3327546.3327739>.
- Hoang-Dung Tran, Diago Manzananas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T Johnson. Star-Based Reachability Analysis of Deep Neural Networks. In *International symposium on formal methods*, pages 670–686. Springer, 2019. doi:10.1007/978-3-030-30942-8_39.
- Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33:1129–1141, 2020a. doi:<https://dl.acm.org/doi/10.5555/3495724.3495820>.
- Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. *arXiv preprint arXiv:2011.13824*, 2020b.
- Alessandro De Palma, Rudy Bunel, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip HS Torr, and M Pawan Kumar. Improved branch and bound for neural network verification via Lagrangian decomposition. *arXiv preprint arXiv:2104.06718*, 2021. doi:<https://doi.org/10.48550/arXiv.2104.06718>.

- Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.
- Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016a.
- Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1928–1937. PMLR, 2016.
- Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.
- Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- Hai Duong, ThanhVu Nguyen, and Matthew Dwyer. A DPLL(T) Framework for Verifying Deep Neural Networks. *arXiv preprint arXiv:2307.10266*, 2024b. doi:10.48550/arXiv.2307.10266.
- Hai Duong, ThanhVu Nguyen, and Matthew Dwyer. Generating and checking dnn verification proofs. *Advances in Neural Information Processing Systems*, 38:65887–65909, 2026d.
- Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. General cutting planes for bound-propagation-based neural network verification. *Proceedings of the 36th International Conference on Neural Information Processing Systems*, 2022. doi:https://dl.acm.org/doi/10.5555/3600270.3600391.
- Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*, 2017.
- Junfeng Wen, Saurabh Kumar, Ramki Gummadi, and Dale Schuurmans. Characterizing the gap between actor-critic and policy gradient. In *International Conference on Machine Learning*, pages 11101–11111. PMLR, 2021.
- Kevin Tan, Wei Fan, and Yuting Wei. Actor-Critics Can Achieve Optimal Sample Efficiency. In *International Conference on Machine Learning*. PMLR, 2025.
- Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.
- Stanley Bak, Changliu Liu, and Taylor Johnson. The Second International verification of Neural Networks Competition (VNN-COMP 2021): Summary and Results. 2021. doi:10.48550/arXiv.2109.00498.
- Hai Duong, ThanhVu Nguyen, and Matthew B Dwyer. Neursat: A high-performance verification tool for deep neural networks. In *International Conference on Computer Aided Verification*, pages 409–423. Springer, 2025.
- Hai Duong and ThanhVu Nguyen. Neursat: Scaling constraint solving for dnn verification (competition contribution). In *International Symposium on AI Verification*, pages 253–259. Springer, 2025.
- Alessandro De Palma, Rudy Bunel, Krishnamurthy Dvijotham, M Pawan Kumar, and Robert Stanforth. Ibp regularization for verified adversarial robustness via branch-and-bound. *arXiv preprint arXiv:2206.14772*, 2022.
- Haoze Wu, Alex Ozdemir, Aleksandar Zeljić, Kyle Julian, Ahmed Irfan, Divya Gopinath, Sadjad Fouladi, Guy Katz, Corina Pasareanu, and Clark Barrett. Parallelization techniques for verifying neural networks. In *2020 Formal Methods in Computer Aided Design (FMCAD)*, pages 128–137, 2020. doi:10.34727/2020/isbn.978-3-85448-042-6_20.
- Duo Zhou, Christopher Brix, Grani A Hanasusanto, and Huan Zhang. Scalable Neural Network Verification with Branch-and-bound Inferred Cutting Planes. *arXiv preprint arXiv:2501.00200*, 2024.
- Dong Xu, Nusrat Jahan Mozumder, Hai Duong, and Matthew B Dwyer. Training for verification: Increasing neuron stability to scale dnn verification. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 24–44. Springer, 2024.
- Jingyue Lu and M. Pawan Kumar. Neural network branching for neural network verification. In *International Conference on Learning Representations*, 2020.
- Vitaly Kurin, Saad Godil, Shimon Whiteson, and Bryan Catanzaro. Can Q-learning with graph networks learn a generalizable branching heuristic for a SAT solver? *Advances in Neural Information Processing Systems*, 33: 9608–9621, 2020.
- Wenxi Wang, Yang Hu, Mohit Tiwari, Sarfraz Khurshid, Kenneth McMillan, and Risto Miikkulainen. NeuroBack: Improving CDCL SAT Solving using Graph Neural Networks. *arXiv preprint arXiv:2110.14053*, 2021b.

- Shumao Zhai and Ning Ge. Learning splitting heuristics in divide-and-conquer SAT solvers with reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025.
- Jan Tönshoff and Martin Grohe. Learning from Algorithm Feedback: One-Shot SAT Solver Guidance with GNNs. *arXiv preprint arXiv:2505.16053*, 2025.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.
- Thanh Le, Van Cuong Le, Bao Thang Ta, and Thi Thanh Binh Huynh. Multitask augmented random search in deep reinforcement learning. *Applied Soft Computing*, 160:111605, 2024.
- Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016b.
- Elisa Y Nakagawa, Milena Guessi, José C Maldonado, Daniel Feitosa, and Flavio Oquendo. Consolidating a process for the design, representation, and evaluation of reference architectures. In *2014 IEEE/IFIP Conference on Software Architecture*, pages 143–152. IEEE, 2014.

A The BaB algorithm

Alg. 3: The BAB_{NNV} algorithm

input : DNN N , property $\phi_{in} \Rightarrow \phi_{out}$
output : unsat if property is valid, otherwise sat

```

1 queue  $\leftarrow \{\emptyset\}$ 
2 while queue  $\neq \emptyset$  do
3    $\sigma \leftarrow \text{SELECT}(\text{queue})$ 
4   if DEDUCE( $N, \phi_{in}, \phi_{out}, \sigma$ ) then
5     ( $cex, v_i$ )  $\leftarrow$  DECIDE( $N, \phi_{in}, \phi_{out}, \sigma$ )
6     if  $cex$  then
7       return sat
8     queue  $\leftarrow$  queue  $\cup \{\sigma \wedge v_i ; \sigma \wedge \bar{v}_i\}$ 
9 return unsat

```

Alg. 3 shows BAB_{NNV} , a reference architecture Nakagawa et al. [2014] for modern DNN verifiers. BAB_{NNV} takes as input a DNN N and a property $\phi_{in} \Rightarrow \phi_{out}$. BAB_{NNV} iterates between two components: DECIDE (branching, line 5) assigns active/inactive status for a neuron, and DEDUCE (bounding, line 4) checks the feasibility of the current subproblem. Both components are critical to verification efficiency, but this work focuses on improving the DECIDE.

B DRL details

We use a variant of soft actor-critic (SAC) Haarnoja et al. [2018] including two Q -networks and target Q -networks supplying bootstrapped value targets, the actor generates a distribution over candidate neurons, which yields policy updates close to deterministic policy-gradient methods but retains the off-policy sample reuse of SAC.

Value objective. The networks are trained to maximize the expected cumulative discounted return, summarized the role of each module via the state-value identity

$$V(s_t) = \mathbb{E}_{a_t \sim \pi} [Q_\phi(s_t, a_t)] \quad (12)$$

Critic update. The critic is trained using temporal difference learning to minimize the prediction error of value estimates:

$$\mathcal{L}_Q(\phi) = \mathbb{E}_{(s_t, a_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\phi(s_t, a_t) - (R(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim P} [V_\phi(s_{t+1})]) \right)^2 \right] \quad (13)$$

where $\mathcal{D}$ is the replay buffer of collected transitions $\langle s_t, a_t, r_t, s_{t+1} \rangle$. The replay buffer enables off-policy learning by storing and reusing past experiences, which improves sample efficiency and stabilizes training. The discount factor $\gamma \in [0, 1]$ controls the trade-off between immediate and future rewards. In particular, $\gamma = 0$ focuses only on immediate rewards, while γ close to 1 encourages the agent to consider long-term consequences.

Actor update. The actor is updated using policy gradients:

$$\mathcal{L}_\pi(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}} [\mathbb{E}_{a_t \sim \pi_\psi} [-Q_\phi(s_t, a_t)]] \quad (14)$$

The actor minimizes this loss to guide action selection toward higher expected returns, as estimated by the critic. The critic learns to estimate action values from collected experience, while the actor uses those estimates to take better actions.

C Training details

C.1 Training setup

Replay, rewards, and buffer capacity. Specifications are listed in Tab. 3. Bounded replay limits memory usage while retaining diverse branching histories from previous problems. When the buffer reaches capacity, the oldest transitions are removed in a first-in, first-out manner, balancing recent and diverse off-policy data. Rewards are normalized by a fixed maximum number of subproblems that an episode may visit, assuring that return scales are comparable across instances with varying search-tree sizes.

Warm-up. During the initial n_{start} environment steps (branching steps summed over episodes), actions are selected using lightweight hand-crafted heuristics (*e.g.*, FSB, RANDOM, etc.) instead of the learned policy. This approach guarantees that early replay can be contained. Optimization may also begin once the buffer is at least half full, even if n_{start} hasn't been reached. After warm-up, letting g index environment steps, we still take a heuristic action occasionally (*e.g.*, every 5 steps). Otherwise, we follow the actor.

Optimization choices (Tab. 3). The actor and critic use learning rates on the order of 10^{-4} , a scale commonly employed in off-policy actor-critic methods for long episodes. The discount factor γ is set to 0.99, making sure that future branching depth is not excessively discounted relative to early decisions. Minibatches contain a few hundred transitions. Gradient accumulation across several draws per update approximates a larger batch and stabilizes the critic targets under GPU memory restrictions. Multiple critic (Bellman) updates are applied before each actor update, allowing Q -values to more closely track the evolving policy. Twin Q -networks and pessimistic targets are used to mitigate overestimation in value backups. Target networks are updated using Polyak averaging with a coefficient $\tau = 10^{-4}$, resulting in slowly moving bootstrap targets.

Network architecture. The actor and critic share the same observations (§5.1) but use separate PointerNet-based networks Bello et al. [2016a]. Each network applies linear projections to raw and embedded neuron features, an LSTM encoder and decoder, n_{glimpse} attention glimpses, and logits clipped to $[\pm 10]$ before applying softmax over admissible neurons. The critic outputs one scalar per branch-child pair using the dual-branch observation (§5.1).

C.2 DRL training progress

Fig. 6 shows actor/critic losses alongside training; both plateau without divergence after roughly 10^6 environment steps in our recorded run. Optional checks, such as periodically running a fixed verification suite to measure branch counts during training, provide a closer proxy to deployment and match our checkpoint schedule. Training on a single GPU for approximately 10^5 verification episodes (batch size 512 with gradient accumulation 8) requires several days. After training, model weights are frozen and shared across all test instances without per-network fine-tuning. Wall-clock time comparisons under competition time budgets are reported in the main text alongside branch counts.

Tab. 3: Hyperparameters for RL training and for GCN embedding pretraining used in our experiments. The episode count equals the main paper’s (10^5 unless stated otherwise).

Hyperparameter	Value
<i>RL training (Actor and Critic)</i>	
Optimizer	AdamW
Actor learning rate	4×10^{-4}
Critic learning rate	4×10^{-4}
Discount factor γ	0.99
Replay buffer capacity	10^5 transitions
Minibatch size	512
Gradient accumulation	8 minibatches per gradient step
Critic updates per policy update	2
Warm-up step n_{start}	10^3 steps
Soft target update τ	10^{-4}
Target network update	1 step
Training episodes	10^5
Checkpoint period	10^4 steps
<i>GCN embedding (before RL training)</i>	
Architecture	2-layer GCN, hidden width 128, ReLU
Node feature	4 (lower bound, upper bound, bias, unstable mask)
Supervision	MSE on FSB scores of unstable neurons
Optimizer	Adam
Learning rate	3×10^{-4}
Epochs	5
Mini-batch size	512

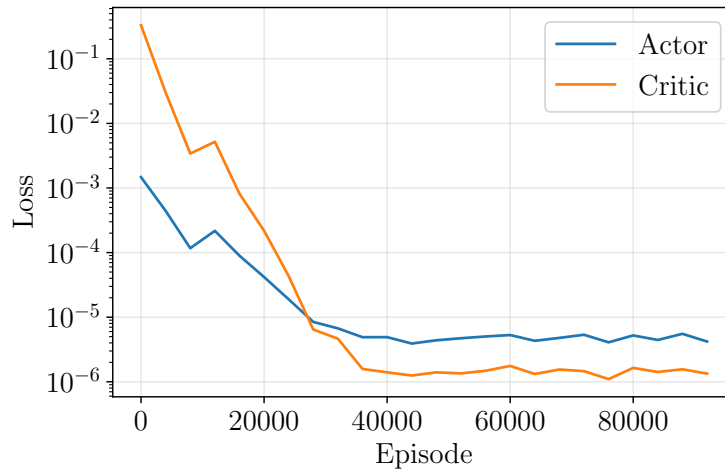


Fig. 6: Training loss over time.