Verifying Neural Networks with Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

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 [1] or medicine [2]. 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 [3]. Dozens of DNN verifiers have been reported in the literature, and a yearly competition has documented advances in their capabilities [4, 5, 6].

As with traditional software verification, DNN verification is often framed as a satisfiability problem and also suffers from scalability challenges [7]. To tackle the large search space of DNN verification, modern verifiers [8, 9, 10, 11] often employ the Branch-and-Bound (BaB) [12] 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 [13, 14, 15, 16, 17, 18, 19].

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 [12] and FSB [20] 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.

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

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 [21] and trains a network [22] via DRL [23, 24] 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 [5, 6] 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 [25]. 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

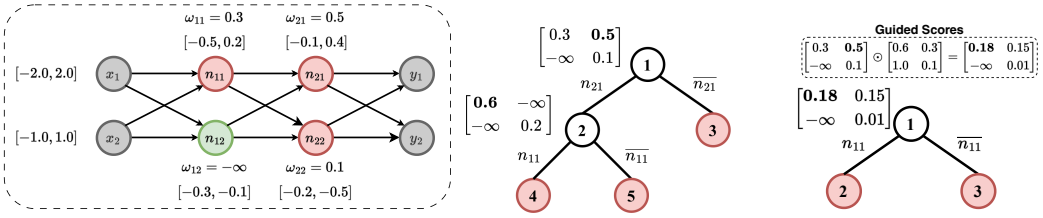
DNN verification. Given a DNN N and a property ϕ, the *DNN verification problem* asks if ϕ is a valid property of N. Typically, ϕ is a formula of the form $\phi_{in} \Rightarrow \phi_{out}$, where ϕ_{in} is a property over the inputs of N and ϕ_{out} is a property over the outputs of N. Modern techniques often treat the DNN verification as a *satisfiability* problem [9, 8, 26, 11, 10]. 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.

Modern DNN verifiers often follow a branch-and-bound (BaB) framework [12], shown in Alg. 3 in Apdx. 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 his relatively under-explored compared to DEDUCE (e.g., designing new abstract domains [16, 15, 17, 14, 18, 19, 27, 10, 8]).

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 [20] 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,



(a) A DNN with neuron bounds computed by an abstraction and heuristic scores ω_{ij} computed by FSB.

(b) A search tree with FSB

(c) A search tree with RSB

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

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 [24], 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 Apdx. B for a detailed actor and critic objectives.

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:

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

BAB$_{\text{NNV}}$ computes a bound estimation of hidden neurons (*e.g.*, using an abstraction as LIRPA [18]) 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$_{\text{NNV}}$ splits (branches) each unstable neuron into positive (active) and negative (inactive) branches.

SOTA BAB$_{\text{NNV}}$ employs various branching heuristics to guide the search, *e.g.*, FSB [20]. 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$_{\text{NNV}}$ with FSB, a popular heuristic used by major DNN verifiers including $\alpha\beta$-CROWN [9], GCP-CROWN [27] and NEURALSAT [10].

First, BAB$_{\text{NNV}}$ 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 $\overline{n_{21}}$ while the positive branch is still undecided. Next, BAB$_{\text{NNV}}$ 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$_{\text{NNV}}$ then can prove both branches and completes the verification. In total, BAB$_{\text{NNV}}$ 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):

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

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:

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

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

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

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 $\overline{n_{11}}$ 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 [21], 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:

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

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

where n_{ij} and $\overline{n_{ij}}$ 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 $n^* = \text{argmax}_{ij}\, \omega_{ij}$, 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:

$$n^* = \underset{i,j}{\text{argmax}}\ \omega_{ij} \odot a_{ij} \tag{7}$$

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:

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

where $\langle N, \phi_{\text{in}}, \phi_{\text{out}}, \sigma \rangle \leftarrow \mathbf{s}$, 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

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

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 4.1 (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. □

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).

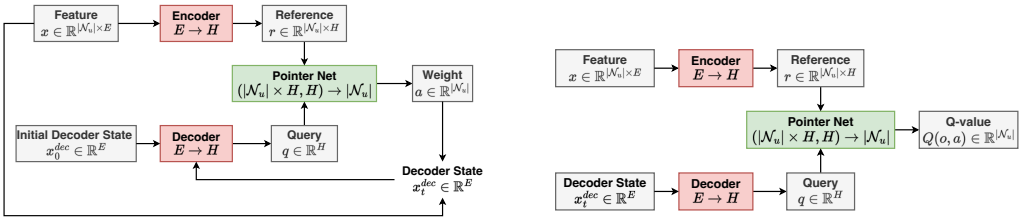
5.1 Observation construction

We train a Graph Convolution Networks (GCN) [28] 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 [24] framework for its sample efficiency [29, 30], which is important since each verification episode is computationally expensive [7]. Both actor $\pi_\psi(\mathbf{a}|\mathbf{o})$ and critics



(a) Actor architecture (b) Critic architecture

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

$Q_\phi(\mathbf{o}, \mathbf{a})$ use PointerNet [22] 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.

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 [24] to capture both successor subproblems:

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

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:

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

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:

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

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

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**

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 branching[1], we applied a two-stage filter using: (i) adversarial attack PGD [31] to remove SAT instances, and (ii) abstraction LIRPA [18] 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.

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 [20] and 200 Sigmoid-based [6] instances, whose networks are not used in training, for generalization.

Baselines. We compare RSB against (i) FSB [20], which estimates bound improvements via Lagrangian Decomposition and is used in $\alpha\beta$-CROWN [9, 27] (ranked 1st in VNN-COMPs—the annual DNN verification competition to compare different approaches and showcase the latest advances in the field [32, 4, 6, 5]); and NEURALSAT [10] (2nd in VNN-COMP'24 and '25); (ii) UPB [33], a fast approximation of FSB; and (iii) POLARITY [34], which uses bound differences directly and is used in MARABOU [11] (ranked 2nd in VNN-COMP'23). All methods are integrated into NEURALSAT for a fair comparison.

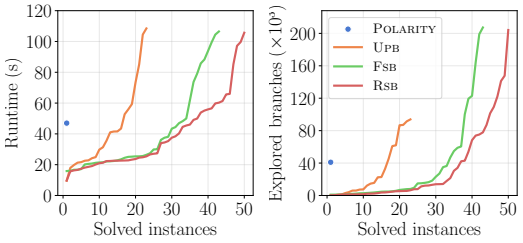
Experimental setup. Our experiments were conducted on a Linux machine with an AMD Ryzen Threadripper 32-Core, 128GB RAM, and an NVIDIA GeForce RTX 4090, 24GB VRAM. We use timeout 120s per instance, which follows the rules in recent VNN-COMPs [5, 6] (6h per benchmark).

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.

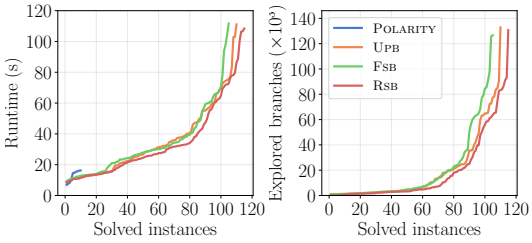
On FNN, RSB verifies 50 instances compared to 43 for FSB (16% more) while reducing total branch exploration by 72% (1.86M vs. 6.65M). On Sigmoid, the gains are even larger: 43 vs. 34 solved

[1] In fact, out of 2700 instances in regular track of VNN-COMP'25 [6], 92% problems (2473/2700) were solved without branching or very few branching steps.



(a) Total runtime. (b) Explored branches.

Fig. 3: Performances on seen networks.



(a) Total runtime. (b) Explored branches.

Fig. 4: Performances on unseen networks.

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 [9], GCP-CROWN [27], and NEURALSAT [10]. 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.

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.

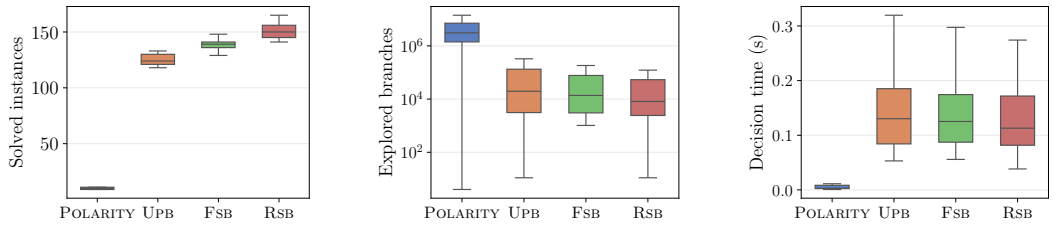
RSB's line (red) extends furthest on the x-axis (50 instances) and runs below FSB throughout both runtimes (Fig. 3a) and explored branches (Fig. 3b), which shows that RSB solves more instances while consistently requiring fewer branches or runtimes. As problem difficulty increases, the gap between RSB 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.



(a) Solved instances. (b) Explored branches. (c) Decision time.

Fig. 5: Statistical distributions of computational metrics.

Fig. 5c shows that RSB's per-step decision time (∼0.11s) is lower than FSB and UPB (∼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 [14], zonotopes [16], polytopes [15, 13, 18], and star sets [17] are widely used by leading verifiers [4, 5, 6]. Bound tightening is further strengthened by multi-neuron relaxations [8], cutting planes [27, 35], and neuron stability [10]. RSB focuses on the branching decision and complements these abstract domains.

FSB [20] is the SOTA heuristic that estimates bound improvements via Lagrangian Decomposition, outperforming BABSR, POLARITY, and GNN heuristic [36]. FSB estimates per-neuron branching scores using tighter bounds from the dual problem, making it a strong greedy heuristic. UPB [33] is an approximation of FSB [20] by reusing dual variables computed during the bounding steps [9]. BABSR [12] uses an abstraction to approximate bound improvements per neuron while POLARITY [34] selects neurons based solely on their bounds. GNN heuristic [36] 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 [37, 38, 39] require a separate GNN forward pass at each decision step, incurring significant overhead. The one-shot guidance approach [40] 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 [12, 20] 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 [40] (§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 [41, 24, 42] and training data grows, performance can continue to improve. Future work includes extending RSB to input-space branching [14] for low-dimensional inputs and online adaptation during verification [43] to enhance the performance of the learned policy on hard instances.

Potential negative societal impact. This work can reveal flaws in sensitive DNN applications. Such information could be exploited for malicious purposes. However, it also enables developers to identify and fix vulnerabilities before deployment.

References

[1] H. Shao, L. Wang, R. Chen, H. Li, and Y. Liu, "Safety-enhanced autonomous driving using interpretable sensor fusion transformer," in *Conference on Robot Learning*, pp. 726–737, PMLR, 2023.

[2] M. X. Morris, A. Rajesh, M. Asaad, A. Hassan, R. Saadoun, and C. E. Butler, "Deep learning applications in surgery: Current uses and future directions," *The American Surgeon*, vol. 89, no. 1, pp. 36–42, 2023.

[3] D. Shriver, S. Elbaum, and M. B. Dwyer, "Reducing DNN Properties to Enable Falsification with Adversarial Attacks," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 275–287, IEEE, 2021.

[4] C. Brix, M. N. Müller, S. Bak, T. T. Johnson, and C. Liu, "First three years of the International Verification of Neural Networks Competition (VNN-COMP)," *International Journal on Software Tools for Technology Transfer*, pp. 1–11, 2023.

[5] C. Brix, S. Bak, T. T. Johnson, and H. Wu, "The Fifth International Verification of Neural Networks Competition (VNN-COMP 2024): Summary and results," *arXiv preprint arXiv:2412.19985*, 2024.

[6] K. Kaulen, T. Ladner, S. Bak, C. Brix, H. Duong, T. Flinkow, T. T. Johnson, L. Koller, E. Manino, T. H. Nguyen, *et al.*, "The 6th International Verification of Neural Networks Competition (VNN-COMP 2025): Summary and Results," *arXiv preprint arXiv:2512.19007*, 2025.

[7] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *International Conference on Computer Aided Verification*, pp. 97–117, Springer, 2017.

[8] C. Ferrari, M. N. Mueller, N. Jovanović, and M. Vechev, "Complete Verification via Multi-Neuron Relaxation Guided Branch-and-Bound," in *International Conference on Learning Representations*, 2022.

[9] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. 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*, vol. 34, pp. 29909–29921, 2021.

[10] H. Duong, D. Xu, T. Nguyen, and M. B. Dwyer, "Harnessing neuron stability to improve DNN verification," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 859–881, 2024.

[11] H. Wu, O. Isac, A. Zeljić, T. Tagomori, M. Daggitt, W. Kokke, I. Refaeli, G. Amir, K. Julian, S. Bassan, *et al.*, "Marabou 2.0: a versatile formal analyzer of neural networks," in *International Conference on Computer Aided Verification*, pp. 249–264, Springer, 2024.

[12] R. Bunel, J. Lu, I. Turkaslan, P. H. Torr, P. Kohli, and M. P. Kumar, "Branch and bound for piecewise linear neural network verification," *Journal of Machine Learning Research*, vol. 21, no. 42, pp. 1–39, 2020.

[13] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[14] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1599–1614, 2018.

[15] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–30, 2019.

[16] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and effective robustness certification," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[17] H.-D. Tran, D. Manzanas Lopez, P. Musau, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson, "Star-Based Reachability Analysis of Deep Neural Networks," in *International symposium on formal methods*, pp. 670–686, Springer, 2019.

[18] K. Xu, Z. Shi, H. Zhang, Y. Wang, K.-W. Chang, M. Huang, B. Kailkhura, X. Lin, and C.-J. Hsieh, "Automatic perturbation analysis for scalable certified robustness and beyond," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1129–1141, 2020.

[19] K. Xu, H. Zhang, S. Wang, Y. Wang, S. Jana, X. Lin, and C.-J. Hsieh, "Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers," *arXiv preprint arXiv:2011.13824*, 2020.

[20] A. De Palma, R. Bunel, A. Desmaison, K. Dvijotham, P. Kohli, P. H. Torr, and M. P. Kumar, "Improved branch and bound for neural network verification via Lagrangian decomposition," *arXiv preprint arXiv:2104.06718*, 2021.

[21] R. S. Sutton, A. G. Barto, *et al.*, *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[22] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016.

[23] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning*, pp. 1928–1937, PMLR, 2016.

[24] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning*, pp. 1861–1870, PMLR, 2018.

[25] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[26] H. Duong, T. Nguyen, and M. Dwyer, "A DPLL(T) Framework for Verifying Deep Neural Networks," *arXiv preprint arXiv:2307.10266*, 2024.

[27] H. Zhang, S. Wang, K. Xu, L. Li, B. Li, S. Jana, C.-J. Hsieh, and J. Z. Kolter, "General cutting planes for bound-propagation-based neural network verification," *Proceedings of the 36th International Conference on Neural Information Processing Systems*, 2022.

[28] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *International Conference on Learning Representations*, 2017.

[29] J. Wen, S. Kumar, R. Gummadi, and D. Schuurmans, "Characterizing the gap between actor-critic and policy gradient," in *International Conference on Machine Learning*, pp. 11101–11111, PMLR, 2021.

[30] K. Tan, W. Fan, and Y. Wei, "Actor-Critics Can Achieve Optimal Sample Efficiency," in *International Conference on Machine Learning*, PMLR, 2025.

[31] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[32] S. Bak, C. Liu, and T. Johnson, "The Second International verification of Neural Networks Competition (VNN-COMP 2021): Summary and Results," 2021.

[33] A. De Palma, R. Bunel, K. Dvijotham, M. P. Kumar, and R. Stanforth, "Ibp regularization for verified adversarial robustness via branch-and-bound," *arXiv preprint arXiv:2206.14772*, 2022.

[34] H. Wu, A. Ozdemir, A. Zeljić, K. Julian, A. Irfan, D. Gopinath, S. Fouladi, G. Katz, C. Pasareanu, and C. Barrett, "Parallelization techniques for verifying neural networks," in *2020 Formal Methods in Computer Aided Design (FMCAD)*, pp. 128–137, 2020.

[35] D. Zhou, C. Brix, G. A. Hanasusanto, and H. Zhang, "Scalable Neural Network Verification with Branch-and-bound Inferred Cutting Planes," *arXiv preprint arXiv:2501.00200*, 2024.

[36] J. Lu and M. P. Kumar, "Neural network branching for neural network verification," in *International Conference on Learning Representations*, 2020.

[37] V. Kurin, S. Godil, S. Whiteson, and B. Catanzaro, "Can Q-learning with graph networks learn a generalizable branching heuristic for a SAT solver?," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9608–9621, 2020.

[38] W. Wang, Y. Hu, M. Tiwari, S. Khurshid, K. McMillan, and R. Miikkulainen, "NeuroBack: Improving CDCL SAT Solving using Graph Neural Networks," *arXiv preprint arXiv:2110.14053*, 2021.

[39] S. Zhai and N. Ge, "Learning splitting heuristics in divide-and-conquer SAT solvers with reinforcement learning," in *The Thirteenth International Conference on Learning Representations*, 2025.

[40] J. Tönshoff and M. Grohe, "Learning from Algorithm Feedback: One-Shot SAT Solver Guidance with GNNs," *arXiv preprint arXiv:2505.16053*, 2025.

[41] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[42] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.

[43] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016.

[44] E. Y. Nakagawa, M. Guessi, J. C. Maldonado, D. Feitosa, and F. Oquendo, "Consolidating a process for the design, representation, and evaluation of reference architectures," in *2014 IEEE/IFIP Conference on Software Architecture*, pp. 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 $\sigma \leftarrow \text{SELECT}(queue)$
4 **if** $\text{DEDUCE}(N, \phi_{in}, \phi_{out}, \sigma)$ **then**
5 $(cex, v_i) \leftarrow \text{DECIDE}(N, \phi_{in}, \phi_{out}, \sigma)$
6 **if** cex **then**
7 **return** *sat*
8 $queue \leftarrow queue \cup \{\sigma \wedge v_i \; ; \; \sigma \wedge \overline{v_i}\}$
9 **return** *unsat*

Alg. 3 shows BAB_{NNV}, a reference architecture [44] 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) [24] 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

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

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

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

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:

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

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

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**
RL training (Actor and Critic)	
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
GCN embedding (before RL training)	
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

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 [22]. 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).

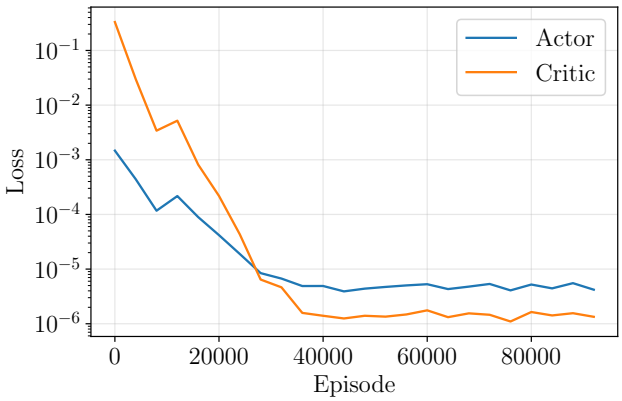


Fig. 6: Training loss over time.

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.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract mentioned "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%.", which is included in §6.2

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: In the conclusion (§8), we discuss three key limitations: (i) the need to continue improving the model as DRL algorithms and training data scale, (ii) the restriction to neuron-space branching which does not yet extend to input-space branching for low-dimensional inputs, and (iii) the absence of online adaptation during verification to handle hard instances.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The soundness and completeness of RSB follow directly from the underlying BAB_{NNV}: RSB only modifies neuron selection order without altering the exhaustive BaB search logic, so the verifier's existing guarantees are preserved (see Thm. 4.1).

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The experimental setup (§6.1) fully describes the dataset, training setup, verification problems, baselines, evaluation metrics, and hardware needed to reproduce the main evaluation results. Apdx. C further details hyperparameter choices and training procedure for RSB.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code is available at https://anonymous.4open.science/r/rsb_branching.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Training and test details including hyperparameters, optimizer, and data splits are provided in §6.1 and Apdx. C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Statistical significance is assessed via box plots (Fig. 5) over five runs with different random seeds for benchmark generation, reporting median and interquartile range of solved instances and branch counts.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Hardware is specified in §6.1: experiments ran on a Linux machine with an AMD Ryzen Threadripper 32-Core processor, 128 GB RAM, and an NVIDIA GeForce RTX 4090 (24 GB VRAM).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The paper conforms with the NeurIPS Code of Ethics. As required, we transparently communicate known consequences of the research via the **Potential negative societal impact** section: " This work can reveal flaws in sensitive DNN applications. Such information could be exploited for malicious purposes. However, it also enables developers to identify and fix vulnerabilities before deployment."

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: The broader impact statement discusses both sides: positively, RSB helps developers identify and fix vulnerabilities in safety-critical DNN applications before deployment; negatively, revealing flaws in sensitive DNN applications could be exploited for malicious purposes.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper poses no risk for misuse. RSB only reveals the potential existence of vulnerabilities in DNN models; crafting actual adversarial inputs requires leveraging adversarial attack methods, which is not our main contribution.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We cited every tool/method/dataset that we used in the paper.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: The paper does not introduce new assets.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development does not involve LLMs.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.