
Clip-and-Verify: Linear Constraint-Driven Domain Clipping for Accelerating Neural Network Verification

Duo Zhou* Jorge Chavez* Hesun Chen Grani A. Hanasusanto Huan Zhang
University of Illinois Urbana-Champaign *Equal Contribution
{duozhou2,jorgejc2,hesunc2,gah}@illinois.edu, huan@huan-zhang.com

Abstract

State-of-the-art neural network (NN) verifiers demonstrate that applying the branch-and-bound (BaB) procedure with fast bounding techniques plays a key role in tackling many challenging verification properties. In this work, we introduce the *linear constraint-driven clipping* framework, a class of scalable and efficient methods designed to enhance the efficacy of NN verifiers. Under this framework, we develop two novel algorithms that efficiently utilize linear constraints to 1) reduce portions of the input space that are either verified or irrelevant to a subproblem in the context of branch-and-bound, and 2) directly improve intermediate bounds throughout the network. The process novelly leverages linear constraints that often arise from bound propagation methods and is general enough to also incorporate constraints from other sources. It efficiently handles linear constraints using a specialized GPU procedure that can scale to large neural networks without the use of expensive external solvers. Our verification procedure, Clip-and-Verify, consistently tightens bounds across multiple benchmarks and can significantly reduce the number of subproblems handled during BaB. We show that our clipping algorithms can be integrated with BaB-based verifiers such as α, β -CROWN, utilizing either the split constraints in activation-space BaB or the output constraints that denote the unverified input space. We demonstrate the effectiveness of our procedure on a broad range of benchmarks where, in some instances, we witness a 96% reduction in the number of subproblems during branch-and-bound, and also achieve state-of-the-art verified accuracy across multiple benchmarks. Clip-and-Verify is part of the α, β -CROWN verifier, **the VNN-COMP 2025 winner**. Code available at https://github.com/Verified-Intelligence/Clip_and_Verify.

1 Introduction

The neural network (NN) verification problem is imperative in mission-critical applications [70, 63, 58, 16, 75, 72] where formally proving properties such as safety and robustness over a specified input domain is essential. Recent approaches make neural network verification more tractable by relaxing the original non-convex problem into convex formulations that are amenable to linear programming (LP) [25, 68], semidefinite programming (SDP) [12, 26, 50, 40, 18, 19], and bound propagation-based solvers [77, 29, 55, 67, 24, 66, 64]. These convex relaxations can be further strengthened by tightening single-neuron relaxations through convex geometric analysis [55], constructing convex-hull approximations to capture multi-input dependencies [48], and introducing cutting planes that encode inter-neuron dependencies [76, 79]. To handle properties that cannot be certified by a single relaxation, state-of-the-art verifiers couple bound propagation methods with the branch-and-bound (BaB) paradigm [14, 66, 53, 13] as this technique can be efficiently parallelized and scaled on GPUs. Linear bound propagation methods such as CROWN [77] recursively compute bounds on the activations of each layer, referred to as *intermediate bounds*, which serve as critical building blocks for determining the tightness of the overall relaxation and for guiding the BaB search.

While BaB and cutting-plane techniques can further refine the relaxation by incorporating additional constraints at the final layer, they cannot *directly* and *efficiently* improve the intermediate bounds themselves. In BaB, where the number of subproblems grows exponentially, loose intermediate bounds weaken the relaxation, resulting in deeper branching and longer verification times. Although algorithms such as β -CROWN [66] theoretically support optimizing bounds at intermediate layers, doing so in practice is prohibitively expensive as the number of hidden neurons typically outnumber the output neurons used for property verification by several orders of magnitude. Consequently, updating bounds for all intermediate layers introduces significant computational overhead that far outweighs the gains brought from their tighter convex relaxations. As a result, existing implementations fix the global intermediate bounds computed at initialization (e.g., via α -CROWN [74]) and focus their optimization efforts solely at the final layer. While this design choice preserves scalability, it limits the ability to tighten the relaxation throughout the network, motivating methods that can refine intermediate bounds both *effectively* and *efficiently*.

To this end, we introduce **Clip-and-Verify**, a verification pipeline designed to enhance NN verifiers by opportunistically refining bounds at **any layer with minimal computational overhead**. Our core insight is that the bounding planes generated by linear bound propagation at all layers naturally align with our pipeline, enabling us to exploit their geometry to eliminate infeasible regions of the input domain and prune redundant subproblems early in verification. We formalize the task of tightening any layer’s bounds as the objective of our linear constraint-driven clipping framework, and we propose two novel algorithms: complete clipping, which directly optimizes neurons’ bounds via a specialized coordinate ascent procedure, and relaxed clipping, which refines the input domain to enhance the intermediate relaxations and consequently improves the NN’s bounds. An intuitive illustration of this refinement is given in Figure 1, and our main contributions are as follows:

- We propose two specialized GPU algorithms within our novel *linear constraint-driven clipping* framework for tightening bounds at **any layer**, preserving the scalability of state-of-the-art NN verifiers without relying on external solvers. *Relaxed clipping* optimizes the input domain bounds as a proxy, offering good improvements with little costs, while *complete clipping* employs a customized coordinate ascent solver to directly refine the bounds at **every layer** of the NN.
- We show that linear bound propagation methods produce linear constraints that can be obtained “for free” during both input and activation BaB procedures. By leveraging these cheaply available constraints and integrating our two efficient clipping algorithms, we introduce **Clip-and-Verify**, a verification pipeline that tightens the all neurons bounds with minimal computational overhead.
- Across a large number of benchmarks from the Verification of Neural Networks Competition (VNN-COMP) [11, 10] and existing literature, we demonstrate that our Clip-and-Verify framework is capable of reducing the number of BaB subproblems by as much as 96% and consistently verifying more properties on benchmarks from the Verification of Neural Networks Competition.

2 Preliminaries

The NN Verification Problem. Given some input $\mathbf{x}$ belonging to the set $\mathcal{X}$, and a feed-forward network $f(\cdot)$ with general activation functions, the goal of NN verification can be formulated as verifying $f(\mathbf{x}) \geq 0$ for all inputs in $\mathcal{X}$. One manner of verifying this property involves solving $\min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x})$, which is challenging due to the non-convexity of the NN and is generally NP-complete [35]. On the other hand, convex-relaxation algorithms compute a sound, approximate lower bound, $\underline{f}(\mathbf{x})$, to the network’s true minimum such that when $\underline{f}(\mathbf{x}) > 0$, the property is sufficiently verified, otherwise the problem is unknown without further refinement or falsification.

Bound Propagation. Linear bound propagation methods [56, 55, 64, 5, 77] approximate neuron bounds layer by layer by relaxing the nonlinearities of activation functions, making these techniques a fast and popular approach for NN verification. For an L -layered, feedforward network, the bounds for the j^{th} neuron at the i^{th} layer may be expressed as:

$$\underline{z}_j^{(i)} := \min_{\mathbf{x} \in \mathcal{X}} \underline{\mathbf{A}}_j^{(i)\top} \mathbf{x} + \underline{\mathbf{c}}_j^{(i)} \leq z_j^{(i)}, \quad \bar{z}_j^{(i)} := \max_{\mathbf{x} \in \mathcal{X}} \bar{\mathbf{A}}_j^{(i)\top} \mathbf{x} + \bar{\mathbf{c}}_j^{(i)} \geq z_j^{(i)} \quad (1)$$

When $\mathcal{X}$ is an ℓ_∞ box (i.e. $\{\mathbf{x} \mid \|\mathbf{x} - \hat{\mathbf{x}}\|_\infty \leq \epsilon\}$), we may “concretize” the lower and upper bounds of the layer using Lemma B.1: $\underline{z}^{(i)} = \underline{\mathbf{A}}^{(i)} \hat{\mathbf{x}} - |\underline{\mathbf{A}}^{(i)}| \epsilon + \underline{\mathbf{c}}^{(i)}$ and $\bar{z}^{(i)} = \bar{\mathbf{A}}^{(i)} \hat{\mathbf{x}} + |\bar{\mathbf{A}}^{(i)}| \epsilon + \bar{\mathbf{c}}^{(i)}$. Once concretized, the post-activation neuron, $\hat{z}_j^{(i)}$, may be bounded (e.g. via Planet relaxation [25]) for

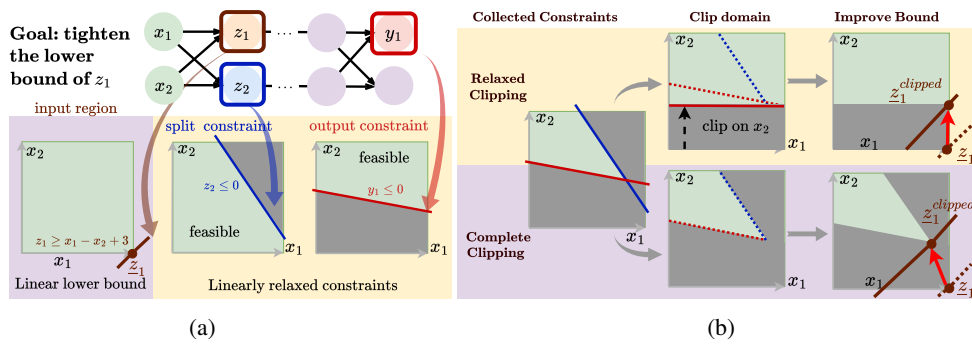


Figure 1: (a) Linear bound propagation produces linear bounds on all neurons w.r.t. the input. These linear bounds are later used as linearly relaxed constraints. In figure (a), the blue and the red lines are used as the linearly relaxed boundary of constraint $z_2 \leq 0$ and $y_1 \leq 0$. Our goal is to further tighten the lower bound of z_1 via these constraints; (b) Linear constraints (e.g. split constraint $z_2 \leq 0$ and output constraint $y_1 \leq 0$) can be applied to shrink the input region and provide tighter bounds. In *Relaxed Clipping*, the feasible region is relaxed to its tightest covering box. In *complete clipping*, the infeasible region is completely cropped off, leaving the exact feasible region to improve bounds on. Both two clipping methods improve the bound to $z_1^{clipped}$, but due to the relaxation nature, Relaxed Clipping yields a looser $z_1^{clipped}$. A detailed numerical example is given in Appendix C.5.

ReLU). The lower/upper bounding planes, $\underline{A}^{(i)}/\overline{A}^{(i)}$ and $\underline{c}^{(i)}/\overline{c}^{(i)}$, are produced via backpropagation, and their definitions are given in Appendix A. The lower bound at the final layer, $\underline{z}^{(L)}$, is used to determine if the problem is verified.

Branch-and-Bound. The BaB paradigm [14, 66, 53, 76, 79, 21, 65, 25, 45] systematically partitions the verification problem into smaller subproblems, $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2$, enabling tighter bounds on each subdomain. BaB can split upon the input space (e.g. axis-aligned constraints) or the activation space (e.g. split activation neurons). See Appendix A.1 for the formal definition and complexity implications. The exponential growth of subdomains can lead to high computational cost but can be mitigated by verifying domains early and often, a direct consequence of *Clip-and-Verify*.

3 Enhancing Neural Network Verification with Clip-and-Verify

3.1 Motivation and Overview

BaB-based verifiers manage complexity by partitioning the input or activation space into tractable subproblems. Within each branch, linear relaxations provide bounds on the network’s neurons which can guide branching decisions and verify properties. Crucially, the constraints introduced at each BaB split implicitly define a tighter feasible input domain and offer opportunities to refine the bounds at any layer. As shown in Fig. 1b, linear constraints can shrink an ℓ_∞ -norm ball during robustness verification, yielding tighter intermediate bounds and enabling more effective search-space pruning.

In large NNs, full linear bound propagation is often limited to an initial pass for efficiency. Alternative methods like using LPs to update intermediate bounds [51] or full re-propagation after each split also prove too costly. Such overhead restricts frequent bound updates in deep NNs or extensive branching.

To fully exploit these opportunities, we propose *Clip-and-Verify*, a pipeline which tightens the input box and re-concretizes intermediate bounds at each BaB node using linear constraints (e.g. activation split and final layer bound), without running a full pass. It contains two algorithms: *Complete Clipping*, which performs a fast coordinate-wise dual search with sorted breakpoints per constraint to obtain near-LP tightening at a fraction of the cost, and *Relaxed Clipping*, which performs per-constraint *exact* tightening for axis-aligned boxes by solving the one-dimensional dual in closed form and then re-concretizing cached linear constraints over the shrunk box.

3.2 Complete Clipping: Optimizing Intermediate Bounds Directly via Linear Constraints

Exact Bound Refinement with a Single Constraint via Optimized Duality. Suppose we wish to improve the lower bound of a linear function $\mathbf{a}^\top \mathbf{x} + c$ over an input domain $\mathcal{X}$, given a new linear

constraint $\mathbf{g}^\top \mathbf{x} + h \leq 0$. The primal optimization problem is:

$$L^* = \min_{\mathbf{x} \in \mathcal{X}} \{ \mathbf{a}^\top \mathbf{x} + c : \mathbf{g}^\top \mathbf{x} + h \leq 0 \} \quad (2)$$

Instead of directly solving this potentially high-dimensional LP, we formulate its Lagrangian dual. The key advantage and insight is that, for a fixed dual variable $\beta \in \mathbb{R}_+$, the inner minimization over $\mathbf{x}$ can be solved analytically for box domains, transforming the problem into a simpler optimization over a **single** dual variable. This leads to the following theorem:

Theorem 3.1 (Exact Bound Refinement under a Single Linear Constraint). *Let $\mathbf{a} \in \mathbb{R}^n$, $c \in \mathbb{R}$, $\mathbf{g} \in \mathbb{R}^n$, $h \in \mathbb{R}$, and the input domain be $\mathcal{X} = \{ \mathbf{x} \mid \hat{\mathbf{x}} - \boldsymbol{\epsilon} \leq \mathbf{x} \leq \hat{\mathbf{x}} + \boldsymbol{\epsilon} \}$. The optimal value L^* of the constrained minimization problem (2) is given by the solution to the dual problem:*

$$L^* = \max_{\beta \in \mathbb{R}_+} \underbrace{(\mathbf{a} + \beta \mathbf{g})^\top \hat{\mathbf{x}} - \sum_{j=1}^n |(\mathbf{a} + \beta \mathbf{g})_j| \epsilon_j + c + \beta h}_{\text{Dual Objective } D(\beta)} \quad (3)$$

$D(\beta)$ in (3) is concave and piecewise-linear in $\beta \in \mathbb{R}_+$. Its maximum L^* and the optimal β^* can be determined exactly and efficiently by identifying its breakpoints (values of β where $(\mathbf{a} + \beta \mathbf{g})_j = 0$ for some $j \in [n]$) and analyzing the super-gradients within the resulting linear segments.

For proof, see Appendix B.2. Theorem 3.1 converts a potentially expensive n -dimensional LP into a 1D concave maximization problem ($D(\beta)$) that can be solved without iterative gradient methods. The process of finding breakpoints and the optimal segment (detailed below and in Algorithm 1 for the multi-constraint case) is highly amenable to efficient computation, making it suitable for refining bounds across many neurons and subdomains **in parallel**. An analogous theorem holds for tightening the upper bound by solving $\max_{\mathbf{x} \in \mathcal{X}} \{ \mathbf{a}^\top \mathbf{x} + c \mid \mathbf{g}^\top \mathbf{x} + h \leq 0 \}$.

Before optimizing Eq. (3), infeasibility can be detected *a priori*: the problem is infeasible iff $\mathbf{g}^\top \hat{\mathbf{x}} + h - \sum_{i=1}^n |g_i| \epsilon_i > 0$. Such a scenario arises when the property being verified imposes constraints unsatisfiable within the current input domain $\mathcal{X}$ (e.g., contradictory ReLU assignments), and may be considered *verified* without further refinement.

Note that our primal optimization problem is mathematically equivalent to a continuous knapsack problem which can be obtained via a change of variables as detailed in Appendix B.3. The breakpoints $\beta_i = -a_i/g_i$ in our dual formulation are identical to the efficiency ratios r_j/s_j used in the standard greedy knapsack algorithm. Thus, our dual-based solver and the greedy knapsack algorithm are equivalent, both finding the provably optimal solution with $\mathcal{O}(n \log n)$ complexity.

Coordinate Ascent for Multiple Constraints When multiple linear constraints $\mathbf{G}\mathbf{x} + \mathbf{h} \leq \mathbf{0}$ (where $\mathbf{G} \in \mathbb{R}^{m \times n}$, $\mathbf{h} \in \mathbb{R}^m$) are available, the dual problem involves optimizing multiple Lagrange multipliers $\boldsymbol{\beta} \in \mathbb{R}_+^m$:

$$L^* = \max_{\boldsymbol{\beta} \in \mathbb{R}_+^m} (\mathbf{a} + \boldsymbol{\beta}^\top \mathbf{G})^\top \hat{\mathbf{x}} - \sum_{j=1}^n |(\mathbf{a} + \boldsymbol{\beta}^\top \mathbf{G})_j| \epsilon_j + c + \boldsymbol{\beta}^\top \mathbf{h} \quad (4)$$

The single constraint problem (3) is easy to solve using Theorem 3.1, thus we can use coordinate ascent to solve (4). Algorithm 1 details a single pass of this coordinate ascent procedure. This iterative approach optimizes one dual variable β_k at a time, keeping others fixed. Each step thus reduces to solving the 1D problem described in Theorem 3.1 (with $\mathbf{a}$ replaced by $\mathbf{a} + \sum_{p \neq k} \beta_p \mathbf{G}_{p,:}$ and $\mathbf{g}$ replaced by $\mathbf{G}_{k,:}$). Since the dual objective remains concave and piecewise-linear along each coordinate β_k , we can efficiently exploit the breakpoint structure for each update. An order dependency is discussed in Appendix B.5. Algorithm 1 is significantly more efficient and scalable than using general-purpose LP solvers, even when those solvers are used as fast heuristics to get a bound rather than converge to optimal. We conducted a detailed comparison in Appendix D.3, integrating a state-of-the-art LP solver (Gurobi) into our BaB framework. The results show that even when running dual simplex with 10-iteration limit, the LP solver was over **880**× **slower** than our GPU-parallelized coordinate ascent (0.0028s vs. 2.47s per round) while achieving comparable bound accuracy (0.00085 vs 0.0007 mean error).

Algorithm 1 Coordinate Ascent with Multiple Constraints

Require: Objective $\mathbf{a}^\top \mathbf{x} + \mathbf{c}$: $\mathbf{a} \in \mathbb{R}^n$, $\mathbf{c} \in \mathbb{R}$; Constraints $\mathbf{G}\mathbf{x} + \mathbf{h} \leq 0$: $\mathbf{G} \in \mathbb{R}^{m \times n}$, $\mathbf{h} \in \mathbb{R}^m$.

- 1: $\boldsymbol{\beta} \leftarrow [0, \dots, 0]^\top$ { Initialize vector of m Lagrange multipliers as zero }
- 2: $\hat{\mathbf{x}} \leftarrow \frac{\bar{\mathbf{x}} + \underline{\mathbf{x}}}{2}$, $\boldsymbol{\epsilon} \leftarrow \frac{\bar{\mathbf{x}} - \underline{\mathbf{x}}}{2}$ { Initialize centroid and the radius of the input hyper-rectangular }
- 3: **for** constraint k in $[m]$ **do**
- 4: $\mathbf{q} \leftarrow -(\mathbf{a} + \boldsymbol{\beta}^\top \mathbf{G}) / \mathbf{G}_{k,:}$ { Calculate the breakpoints }
- 5: $\mathbf{I} \leftarrow \text{argsort}(\mathbf{q})$
- 6: $\mathbf{g} \leftarrow |\mathbf{G}_{k,:}| \odot \boldsymbol{\epsilon}$ { Scale by box half-widths }
- 7: $\mathbf{g}_{\text{sorted}} \leftarrow \mathbf{g}_{\mathbf{I}}$ { Reorder by the same argsort index $\mathbf{I}$ }
- 8: $\mathbf{g}_i^{(-)} \leftarrow -\sum_{j=1}^i (|\mathbf{g}_{\text{sorted}}|_j)$, $j \in [n]$ { (Negative) cumulative sum of sorted breakpoints }
- 9: $\mathbf{g}^{(+)} \leftarrow \mathbf{g}^{(-)} - \mathbf{g}_n^{(-)}$ { Shift $\mathbf{g}^{(-)}$ to its positive range }
- 10: $\nabla \mathbf{g} \leftarrow \mathbf{g}^{(+)} + \mathbf{g}^{(-)} + \mathbf{G}_{k,:}^\top \hat{\mathbf{x}} + \mathbf{h}_k$ { Supergradient; monotone nondecreasing in i }
- 11: $i^* \leftarrow \min\{i \in [n] : \nabla g_i \leq 0\}$ { First sign change }
- 12: $j^* \leftarrow \mathbf{I}_{i^*}$ { Map the i^* -th breakpoint in the sorted order back to the original coordinate index }
- 13: $\boldsymbol{\beta}_k \leftarrow \max\{\mathbf{q}_{j^*}, 0\}$ { Ensure feasibility }
- 14: $l^* \leftarrow -\|\mathbf{a} + \boldsymbol{\beta}^\top \mathbf{G}\|_1 \cdot \boldsymbol{\epsilon} + (\mathbf{a} + \boldsymbol{\beta}^\top \mathbf{G}) \hat{\mathbf{x}} + \boldsymbol{\beta}^\top \mathbf{h} + \mathbf{c}$ { Calculate dual objective }
- Ensure:** l^*

3.3 Relaxed Clipping: Optimizing Intermediate Bounds Indirectly via Input Refinement

Although the coordinate ascent is significantly efficient compared to LP solvers, it is still computationally expensive when there are various subproblems with many unstable neurons and multiple constraints. To address this issue, we propose a more efficient *Relaxed Clipping* algorithm, which **shrinks the input domain** by leveraging linear constraints (e.g., from final-layer outputs or activation branchings) to shrink axis-aligned portions of the input. By adopting a box input domain as a proxy, we avoid repeatedly solving linear programs for each intermediate-layer neuron, and can solve the resulting optimization problem efficiently via the dual norm without relying on external solvers. The remaining task is to determine the clipped box $\mathcal{X}' \subseteq \mathcal{X}$ that best respects the constraints.

Formulating Relaxed Clipping for a Hyper-Rectangle. We assume a hyper-rectangular input region and a set of m linear constraints $\mathbf{A}\mathbf{x} + \mathbf{c} \leq 0$, with $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{c} \in \mathbb{R}^m$. To refine the lower and upper bounds of the input along each input dimension $i \in [n]$, we solve:

$$\underline{x}_i := \min_{\mathbf{x} \in \mathcal{X}} \{x_i \mid \mathbf{A}\mathbf{x} + \mathbf{c} \leq 0\}, \quad \bar{x}_i := \max_{\mathbf{x} \in \mathcal{X}} \{x_i \mid \mathbf{A}\mathbf{x} + \mathbf{c} \leq 0\}. \quad (5)$$

These refined bounds, $\underline{x}_i$ and $\bar{x}_i$, **clip** the original domain $\mathcal{X}$ to reflect only the portion that satisfies all the linear constraints. We derive a simple **closed-form solution** when there is only one linear inequality, $\mathbf{a}^\top \mathbf{x} + c \leq 0$, foregoing the need to solve (5) via LP solvers or gradient-based methods.

Theorem 3.2 (Relaxed clipping under a single constraint). *Let $\mathbf{x} \in \mathcal{X}$ and $\mathbf{a}^\top \mathbf{x} + c \leq 0$ be the sole constraint. For brevity, denote the closed-form solution as $\mathbf{x}_i^{(\text{clip})} = (-\sum_{j \neq i} \{\mathbf{a}_j \hat{\mathbf{x}}_j - |\mathbf{a}_j| \epsilon_j\} - c) / \mathbf{a}_i$. Then, for each coordinate i , the new upper (or lower) bound is updated as follows:*

$$\begin{cases} \bar{x}_i^{(\text{new})} = \min \left\{ \mathbf{x}_i^{(\text{clip})}, \bar{x}_i \right\} & \text{if } \mathbf{a}_i > 0 \\ \underline{x}_i^{(\text{new})} = \max \left\{ \mathbf{x}_i^{(\text{clip})}, \underline{x}_i \right\} & \text{if } \mathbf{a}_i < 0 \\ \text{no change} & \text{otherwise} \end{cases}$$

Appendix B.4 gives a proof showing that, for a single linear constraint $\mathbf{a}^\top \mathbf{x} + c \leq 0$, we can **clip** the box $\mathcal{X}$ to a new box $\mathcal{X}'$ in one pass using the closed-form updates in Theorem 3.2, without any external solvers. The resulting box is the *tightest axis-aligned over-approximation* of the feasible set $\mathcal{X} \cap \{\mathbf{x} : \mathbf{a}^\top \mathbf{x} + c \leq 0\} \subseteq \mathcal{X}' \subseteq \mathcal{X}$, and it is *component-wise tight*: no coordinate bound of $\mathcal{X}'$ can be further refined while remaining a box that still contains the feasible set. The computation costs only $O(n)$ arithmetic operations. For example, re-propagating bounds for a network with 3 intermediate layers (4096, 2048, and 100 neurons respectively) and containing 800-1600 unstable neurons can take 10s per subdomain, versus 0.3s when re-concretizing the intermediate bounds.

This Relaxed Clipping step complements Complete Clipping. Whereas Complete Clipping solves problem (3) for each neuron, Relaxed Clipping tightens the shared input box $\mathcal{X}'$ once, and then

all intermediate bounds are cheaply re-concretized over this tighter box, yielding network-wide bound improvements from a single cheap update. By keeping the shared domain as a box, we avoid polyhedral operations and repeated per-neuron optimizations, substantially improving scalability.

Clipping for Multiple Constraints. When several linear constraints are present, we apply Theorem 3.2 in *parallel* to each constraint in $\mathbf{Ax} + \mathbf{c} \leq 0$. Algorithm 2 outlines this procedure: Given the original box bounds, $\underline{x}$ and $\overline{x}$, we compute its center and radius, $\hat{x}$ and ϵ . Then, for each constraint k and dimension i , we apply Theorem 3.2 independently and in parallel to refine $\underline{x}_i$ and $\overline{x}_i$. After processing all constraints, the resulting clipped bounds are aggregated, and the tightest bounds are selected to form the final clipped domain. This formulation preserves scalability and supports parallelization. We present a sequential variation in Appendix C.2 that foregoes parallelization for further refinement in which the box center and radius are recalculated after each constraint is applied. Nonetheless, we emphasize our current formulation as a core strength of domain clipping, maintaining both efficiency and effectiveness. Similar to direct clipping, Algorithm 2 may identify *infeasibility* by returning clipped bounds where $\underline{x}$ is larger than $\overline{x}$ along some dimension(s).

Algorithm 2 Linear Constraint-Driven Relaxed Clipping (*Parallel*)

Require: $\underline{x}$: Input lower bounds; $\overline{x}$: Input upper bounds; $\mathbf{Ax} \leq \mathbf{c}$: Constraints.

```

1:  $\hat{x} \leftarrow \frac{\overline{x} + \underline{x}}{2}$ ,  $\epsilon \leftarrow \frac{\overline{x} - \underline{x}}{2}$ ,  $\underline{x}^{(\text{clipped})} \leftarrow \underline{x}$ ,  $\overline{x}^{(\text{clipped})} \leftarrow \overline{x}$  { Initialize the original bounds }
2: for each constraint  $k \in \{1, \dots, \text{rows}(\mathbf{A})\}$  do
3:   for each input dimension  $i \in \{1, \dots, \text{cols}(\mathbf{A})\}$  do
4:      $\mathbf{x}_i^{(\text{new})} \leftarrow \frac{-\sum_{j \neq i} \mathbf{A}_{k,j} \hat{x}_j + \sum_{j \neq i} |\mathbf{A}_{k,j}| \epsilon_j - \mathbf{c}_k}{\mathbf{A}_{k,i}}$  { Update  $x$  using Theorem 3.2 }
5:     if  $\mathbf{A}_{k,i} \geq 0$  then
6:        $\overline{x}_i^{(\text{clipped})} \leftarrow \min(\overline{x}_i^{(\text{clipped})}, \mathbf{x}_i^{(\text{new})})$  { Iteratively tighten the upper bound }
7:     else
8:        $\underline{x}_i^{(\text{clipped})} \leftarrow \max(\underline{x}_i^{(\text{clipped})}, \mathbf{x}_i^{(\text{new})})$  { Iteratively tighten the lower bound }
```

Ensure: $\underline{x}^{(\text{clipped})}$, $\overline{x}^{(\text{clipped})}$

We have discussed how to use the linear constraints to tighten bounds, the next step is to find these linear constraints. We can use output constraints or activation split constraints during BaB. The next two sections we will discuss how to find the constraints and incorporate our algorithm into BaB.

3.4 Integrating Clipping into Input BaB Verification

In input BaB, the verifier partitions the network's *input region* (often an ℓ_∞ -box) along axis-aligned splits, generating multiple subdomains $\{\mathcal{X}_i\}_{i=1}^b$. Each subdomain is then fed into a bound-propagation verifier (e.g., CROWN) to obtain linear hyper-planes that are used to lower-bound the *final-layer output* with respect to the input. Existing verifiers typically use these final-layer hyperplanes *only* to compute the bounds and decide whether the *entire* subdomain can be immediately verified or must be further subdivided. It may be the case that the hyperplanes used for computing the final layer bounds are tight enough to verify a subset of the input, but not tight enough to verify the input in its entirety (see Fig. 4 in Appendix). The *key insight* is that we can remove infeasible parts in a subproblem via linear constraints and then consider a partially verified problem in the next iteration of BaB. These already calculated hyperplanes can then be effectively re-used as constraints, making them appropriate for our clipping paradigms.

Using Final-Layer Bounds for Clipping. After a subdomain has been bounded, the final-layer's bounding plane(s) are retained after *concretization* if the subdomain cannot be fully verified. For the verification objective, $f(\mathbf{x}) \geq 0$, it suffices to verify the input region, $\{\mathbf{x} \mid \underline{\mathbf{a}}^{(L)\top} \mathbf{x} + \underline{\mathbf{c}}^{(L)} \geq 0\}$. In this case, the bounding plane acts as a constraint that separates this verified subset from the complimentary subset of the input domain that requires further analysis, i.e., $\{\mathbf{x} \mid \underline{\mathbf{a}}^{(L)\top} \mathbf{x} + \underline{\mathbf{c}}^{(L)} \leq 0\}$. For verification problems with multiple output conditions in the form of $f_1(\mathbf{x}) \geq 0 \wedge f_2(\mathbf{x}) \geq 0 \wedge \dots \wedge f_T(\mathbf{x}) \geq 0$, we can have multiple linear bounds (1), one for each clause, and all of them can be used. Rather than treating each property separately, we jointly collect all relevant final-layer constraints and apply them using Algorithms 1 and 2, accelerating batch verification on a GPU.

Modifications to the Standard Input BaB Procedure. Algorithm 4 in Appendix C.3 outlines our modification to the standard input BaB loop, highlighted in **brown**. Our first key modification after bounding a batch of domains, is that we perform relaxed clipping **after** we split the domains. This **reordering** allows two child domains to inherit their parent’s constraint after axis-aligned split. The inheritance enables a *more tailored domain clipping* procedure for each child, which is often more effective than clipping the parent domain directly. The resulting subdomains are reinserted into the domain list along with their associated constraints. In the subsequent bounding step, we perform complete clipping for the unstable neurons in each layer, where constraints are leveraged to *directly* to tighten the lower/upper bound objectives. When the number of unstable neurons is large, we heuristically select a subset of critical neurons for Complete Clipping; the selection heuristics are detailed in Appendix D.4. Neurons not selected for complete clipping still benefit indirectly, as their concretized bounds are refined due to the clipped domain inherited from the prior BaB iteration.

3.5 Integrating Clipping into Activation BaB Verification

We now demonstrate how our clipping framework naturally extends to branch-and-bound on the activation space. Our method is **activation-agnostic**, as it operates on general linear constraints derived from **any activation** split. For a given neuron $z_j^{(i)}$, BaB can introduce splits of the form $z_j^{(i)} \geq s$ or $z_j^{(i)} \leq s$, where s is a split point ($s = 0$ for a ReLU split)[53]. This inequality can be relaxed by bound propagation algorithms and expressed as a linear constraint on the input, $\mathbf{g}^\top \mathbf{x} + h \leq 0$, which is directly usable by our algorithms. This flexibility allows our framework to apply to networks with various activation functions, as demonstrated by our experiments on Vision Transformer models in Section 4. By assigning certain unstable neurons to either regime, we obtain linear relaxation split constraints that can further optimize the *intermediate bounds*, boosting verification efficacy. A key insight is that **many constraints** from multiple activation assignments can accumulate, potentially tightening intermediate bounds significantly.

Using Linear Constraints in Activation Space for Clipping. We prioritize neurons whose convex envelopes incur the largest relaxation error; for ReLU this typically coincides with “unstable” units (Appendix A.1). At any BaB node with input box $\mathcal{X}_t$, linear bound propagation yields, for each neuron (i, j) ,

$$\underline{\mathbf{A}}_j^{(i)\top} \mathbf{x} + \underline{\mathbf{c}}_j^{(i)} \leq z_j^{(i)}(\mathbf{x}) \leq \overline{\mathbf{A}}_j^{(i)\top} \mathbf{x} + \overline{\mathbf{c}}_j^{(i)} \quad \forall \mathbf{x} \in \mathcal{X}_t. \quad (6)$$

Then we can **validate** of activation-space linear constraints. If a branch assigns $z_j^{(i)} \geq s$, then any feasible $\mathbf{x}$ must satisfy

$$\overline{\mathbf{A}}_j^{(i)\top} \mathbf{x} + \overline{\mathbf{c}}_j^{(i)} \geq s, \quad (7)$$

since $z_j^{(i)}(\mathbf{x}) \leq \overline{\mathbf{A}}_j^{(i)\top} \mathbf{x} + \overline{\mathbf{c}}_j^{(i)}$. Symmetrically, for $z_j^{(i)} \leq s$ we obtain the necessary condition $\underline{\mathbf{A}}_j^{(i)\top} \mathbf{x} + \underline{\mathbf{c}}_j^{(i)} \leq s$. Thus the activation-branching provide *sound linear constraints in the input space* that encode the assigned activation regime.

We can cache and **reuse** the initialized activation branching linear constraints. When bound initialization is executed at a node, we cache $(\underline{\mathbf{A}}, \underline{\mathbf{c}})$ and $(\overline{\mathbf{A}}, \overline{\mathbf{c}})$ for all “unstable” neurons, and further use the constraints based on the branching domains during BaB to do (i) *Relaxed Clipping*: treat (7) (and its lower-bound analogue) as box-consistency constraints and update $\mathcal{X}_t \mapsto \mathcal{X}'_t$ in closed form following Algorithm. 2). (ii) *Complete Clipping*: on a selected set $\mathcal{C}^{(p)}$ of neurons, we directly tighten their per-neuron affine bounds using the same constraints using Algorithm. 1. Neurons not in $\mathcal{C}^{(p)}$ keep their original bounds, but still tighten indirectly when re-concretized over $\mathcal{X}'_t$.

Here, $\mathcal{C}^{(p)}$ is the set of “critical neurons” that are heuristically expected to benefit most from bound refinement. While one could apply our method to every neuron in the network, this may be challenging for networks with high-dimensional hidden layers. To address this, we propose a top-k objective selection heuristic that adaptively selects neurons based on the BaBSR intercept score [13]. This heuristic strategically prioritizes neurons whose refinement is most likely to tighten the overall verification bounds. A detailed justification for this choice, including an ablation study comparing BaBSR to other common heuristics, is provided in Appendix D.4. Neurons not selected by this heuristic retain their standard bounds as computed by CROWN but may still benefit *indirectly* from relaxed clipping. Since relaxed clipping is computationally lightweight and highly scalable, it should always be applied to refine the input domain as effectively as possible.

Table 1: Comparison of different toolkits on a few representative VNN-COMP benchmarks with input BaB. “-” indicates that the benchmark was not supported. Time is calculated as the total time taken to verify verified instances. The number of BaB subproblems by α , β -CROWN is set as the baseline, and the reduction rate is calculated from this number. Complete clipping significantly reduces the number of subproblems during BaB, while relaxed clipping is sometimes faster overall. Reordering generally helps reduce time and subproblems.

	lsnc			acasxu			nn4sys		
Method	time(s)	subproblems	# verified	time(s)	subproblems	# verified	time(s)	subproblems	# verified
nenum* [5, 7]	-	-	-	213.41	-	139	167.55	-	22
Marabou†‡ [36, 71]	-	-	-	1342.03	-	134	151.31	-	24
PyRAT† [30]	90.40	-	15	1484.39	-	137	704.55	-	53
Never2† [30]	-	-	-	1368.78	-	121	-	-	-
NNV† [62]	-	-	-	2631.49	-	70	-	-	-
Cora† [1]	-	-	-	1566.80	-	134	22.88	-	2
NeuralSAT† [22]	-	-	-	1316.85	-	138	-	-	-
α , β -CROWN†	115.27	142,293,985	40	280.51	7,154,387	138	1580.66	4,440,252	194
Clip-and-Verify (Ours)									
Relaxed clipping	99.12	92,402,227 ^{↓35.1%}	40	151.37	3,124,100 ^{↓56.3%}	139	1193.89	2,691,750 ^{↓39.4%}	194
Relaxed + Reorder	98.42	66,412,652 ^{↓53.3%}	40	150.25	2,557,715 ^{↓64.2%}	139	1166.08	2,300,894 ^{↓48.2%}	194
Complete clipping	84.30	5,334,421^{↓96.3%}	40	168.57	1,533,068^{↓78.6%}	139	2846.06	2,141,288^{↓51.8%}	194

Table 2: Performance of Clip-and-Verify on challenging control system verification tasks. Complete clipping is essential for verifying the most difficult properties. The timeout is 3 days (259,200s).

	cartpole		Quadrotor-2D		Quad-2D-Large	
Method	time(s)	subproblems	time(s)	subproblems	time(s)	subproblems
α , β -CROWN (No clipping)	1602	54,260,909	timeout	timeout	timeout	timeout
Clip-and-Verify (Relaxed clipping)	484	16,453,971 ^{↓69.7%}	209,504	2,630,043,050	timeout	timeout
Clip-and-Verify (Complete clipping)	142	2,438,359^{↓95.5%}	78,818	1,112,917,436^{↓57.7%}	104,614	1,472,433,971

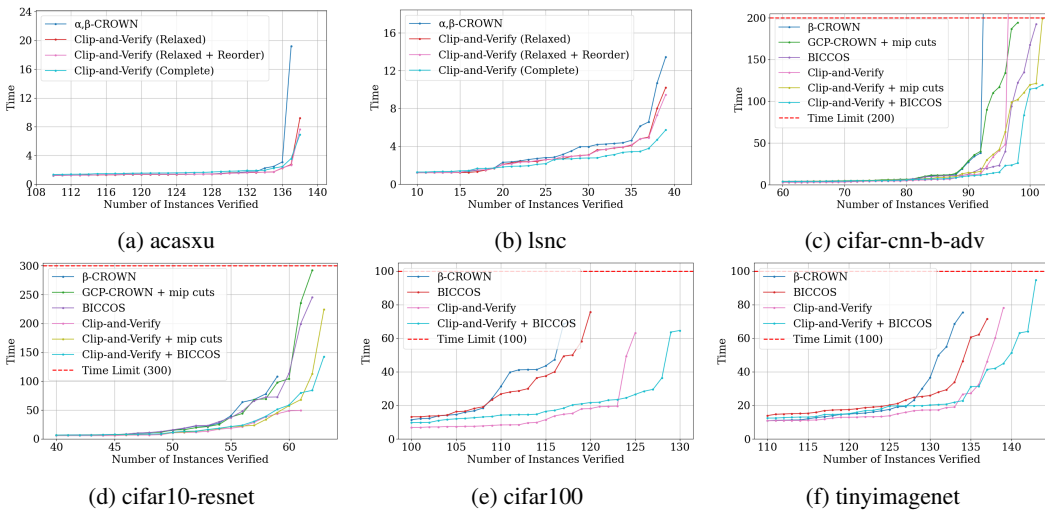


Figure 2: Representative benchmarks visualization. (a) and (b) are input BaB benchmarks with timeout 120s and 100s respectively, (c) and (d) represent medium-size ReLU nets, whereas (e) and (f) are substantially larger. These latter 4 benchmarks utilize ReLU splitting, for which we use complete clipping. Despite their differences in scale, our Clip-and-Verify algorithm demonstrates strong performance on both. Please refer to appendix D.2 for a detailed interpretation.

Modifications to the Standard Activation BaB Procedure. Our activation space BaB Algorithm 5, shown in Appendix C.4, iteratively uses activation split constraints to optimize the intermediate bounds via Complete Clipping (Theorem 3.1). This strategy fully exploits accumulated constraints through complete clipping on intermediate bounds directly, bypassing costly LP solves.

4 Experiments

We evaluate the effectiveness of Clip-and-Verify on several benchmarks from VNN-COMP 2021-2024 [6, 49, 11, 10]. We first demonstrate its benefits on three benchmarks and three hard NN control system problems that are commonly solved with the input BaB procedure, then evaluate our approach on challenging activation space BaB benchmarks in VNN-COMPs, as well as SDP-FO benchmarks

Table 3: Comparison of different toolkits and Clip-and-Verify on VNN-COMP benchmarks with activation split BaB. Results of β -CROWN and BICCOS were from the same hardware of our experiments for a direct comparison; other results are from VNN-COMP reports. “-” indicates the benchmark was not supported.

	oval22		cifar10-resnet		cifar100-2024		tinyimagenet-2024	
Method	time(s)	# verified	time(s)	# verified	time(s)	# verified	time(s)	# verified
nnenum* [5, 7]	630.06	3	-	-	-	-	-	-
ERAN* [47, 46]	233.84	6	24.74	43	-	-	-	-
OVAL* [21, 20]	393.14	11	-	-	-	-	-	-
Venus2 [9, 39]	386.71	17	-	-	-	-	-	-
VeriNet† [32, 33]	73.65	17	8.11	48	-	-	-	-
MN-BaB† [27]	137.13	19	-	-	-	-	-	-
Marabou‡ [36, 71]	5.33	19	40.42	39	-	0	-	0
PyRAT‡ [30]	-	-	-	-	42.38	68	55.64	49
β -CROWN [77, 66]	29.39	20	9.17	60	8.15	119	8.65	135
BICCOS [76, 79]	31.72	25	16.73	63	8.75	121	9.73	138
Clip-and-Verify with β -CROWN	28.15	22	6.06	63	6.59	126	9.48	140
Clip-and-Verify with BICCOS	46.48	27	11.81	64	8.17	131	10.48	144
Upper Bound		29		72		168		157

* Results from VNN-COMP 2021 report [6]. † from VNN-COMP 2022 report [49] ‡ from VNN-COMP 2024 report [10]

Table 4: Verified accuracy (Var.%), avg. per-example verification time (s) on VNN-COMP benchmarks and other commonly used benchmarks. The average time is calculated on verified images only. Clip-and-Verify consistently outperforms all baselines when combined with state-of-the-art BaB verifiers such as BICCOS [79]

Dataset	Model	β -CROWN		GCP-CROWN with MIP cuts		BICCOS		Clip-and-Verify with β -CROWN		Clip-and-Verify with MIP cuts		Clip-and-Verify with BICCOS		Upper bound
		Ver.%	Time (s)	Ver.%	Time(s)	Ver.%	Time(s)	Ver.%	Time(s)	Ver.%	Time(s)	Ver.%	Time(s)	
MNIST	CNN-A-Adv	71.0	4.28	71.5	6.64	76.0	6.91	74.0	3.33	73.5	5.15	76	6.39	76.5
	CNN-A-Adv	45.5	5.26	48.5	4.81	48.5	3.91	45.5	1.35	48.5	3.59	48.5	3.08	50.0
	CNN-A-Adv-4	46.5	1.25	48.5	2.64	48.5	1.55	46.5	0.62	48.5	2.26	48.5	1.31	49.5
	CNN-A-Mix	42.0	4.00	47.5	10.49	48.0	9.00	43.0	4.24	47.5	10.53	48.0	7.45	53.0
	CNN-A-Mix-4	51.0	1.05	55.0	5.94	56.0	9.22	51.0	0.84	55.0	4.44	56.5	5.12	57.5
	CNN-B-Adv	47.0	8.25	49.5	12.16	51.0	10.79	49	7.54	51.5	11.68	51.5	7.68	65.0
	CNN-B-Adv-4	55.0	2.91	58.5	7.01	59.5	4.7	56.5	0.82	60.0	6.67	60.5	5.55	63.5
cifar10-resnet		83.33	9.17	87.5	17.99	87.5	16.73	86.11	6.06	88.89	16.73	88.89	11.80	100.0
oval22		66.66	29.39	83.33	63.53	83.33	31.72	73.33	28.15	90.00	45.52	90.00	46.48	96.67
cifar100-2024		59.5	8.15	-	-	60.5	8.75	63.0	6.59	-	-	65.5	8.17	84.0
tinyimagenet-2024		67.5	8.65	-	-	69.0	9.73	70.0	9.48	-	-	72.0	10.48	78.5
vision-transformer 2024 [53]		59.0	22.04	-	-	-	-	61.0	10.81	-	-	-	-	100.0

introduced in previous studies [19, 66]. Fig. 2 shows the overview of results of selected benchmarks. All results visualization and details about the configuration see Appendix D.1.

Verification Results on Input Split Benchmarks and Hard NN Control Systems. For evaluation, we focus on the following benchmarks from VNN-COMP 23 and VNN-COMP 24: *acasxu*, *lsnc*, and *nn4sys*. we have considered all input benchmarks and these three are the mostly challenging ones when solved using input BaB. We compare our approach to other baseline tools as shown in Table 1. Overall, Clip-and-Verify reduces the number of branches by over 50% and accelerating the verification process. For *lsnc*, we reduced the number of domains by 96%. We further tested our method on three **challenging** verification tasks from a recent study on **provably stable neural network control systems** [75, 41, 42]: *cartpole*, *Quadrotor-2D*, and *Quadrotor-2D-Larger-R0A*. These tasks require certifying Lyapunov-based stability over high-volume state domains, and can not be solved by existing verifiers. The challenge of the verification problem is that we need to verify inside the intersection of a large box and a level set of its Lyapunov function. Detailed settings see Appendix D.1. These instances therefore test whether a verifier can reason about Lyapunov decrease and boundary non-escape over large, physically meaningful state ranges, rather than over small adversarial balls. While general-purpose robustness verifiers are not designed for such volumes, our method targets exactly this regime. As shown in Table 2, baseline methods without clipping fail to solve these problems within the time limit. Both relaxed and complete clipping enable verification, with **complete clipping** demonstrating superior performance by drastically reducing both the number of visited BaB subproblems and the total runtime, turning previously intractable problems into verifiable ones.

Verification Results on General Activation Split Benchmarks Shown in Table 3, we further compare the proposed methods against a wide range of existing neural network verification toolkits on six challenging VNN-COMP benchmarks: *oval22*, *cifar10-resnet*, *cifar100-2024*, *tinyimagenet-2024*, and *vit-2024*. Each method is evaluated in terms of average runtime (seconds) and the number of verified properties (# verified). A dash (-) denotes that the corresponding tool

was not applicable or did not support that particular benchmark. Clip-and-Verify with BICCOS attains state-of-the-art verification coverage on the evaluated VNN-COMP benchmarks, pushing closer to the theoretical upper bound in terms of the number of properties verified. When combined with BICCOS, whose tight bounding routines complement Clip-and-Verify’s efficient splitting, coverage reaches to 131 and 144, underscoring the method’s broad applicability and scalability. Besides of the ReLU nets, we further evaluated our method on a **Vision Transformer** (ViT) model [53], which contains **non-ReLU** layers such as **Softmax**. Verifying such architectures is challenging due to the complex, non-linear constraints introduced by these operators. The model tested has approximately 76k parameters and an input dimension of 3072. Tables 4, 6 show that Clip-and-Verify significantly outperforms the baseline, verifying more properties in about half the time and with nearly 45% fewer subproblems in ViT. This demonstrates the applicability of our framework on general networks.

Ablation Studies. We compare the BaB baseline without clipping, *Relaxed Clipping* only, and the full pipeline (Relaxed + Complete) in Table 1. We further vary the comparison of complete clipping and LP solvers in Section D.3, and *scoring rule* for choosing critical neurons (e.g., BaBSR intercept, envelope gap, bound width) in Complete Clipping in Section D.4. For each configuration we report time and visited subproblems under identical branching/timeout settings in Table 4 and Table 6; detailed breakdowns, per-model trends, and ablation protocols are in Appendix D.2.

5 Related Work

Verification becomes easier when we tighten the *base relaxations* that bound intermediate layers and *contract the input region*, so that a subsequent branch-and-bound (BaB) search explores fewer and easier subproblems. BaB frameworks unify many complete verifiers and show that stronger node bounds reduce tree size [14, 13, 55, 64, 77, 5]. Subsequent improvements include faster dual/active-set solvers and better branching [20, 66, 27]. Our clipping steps make each node *strictly easier* (smaller box and tighter intermediate relaxations) without spawning additional subproblems, and are therefore complementary to any branching heuristic. Our method contracts the box domain using linear constraints and injects the same constraints to tighten intermediate affine bounds, thus improving the node-wise base problem before any additional branching is introduced.

For tighter base relaxations, linear-relaxation-based bound propagation (e.g., CROWN [77] and DeepPoly [56]) provide fast base bounds. Beyond this, early single-neuron triangle relaxations such as PLANET [25] and the convex adversarial polytope/Wong–Kolter line [69] underpin modern propagation methods; symbolic/abstract-interpretation variants (DeepZ [55]/AI2 [29]/Neurify [64]/OSIP [31]) further improve hidden-layer bounds in practice. Multi-neuron relaxations overcome single-neuron limits by coupling activations [54, 2, 48, 57]. Beyond k-ReLU and PRIMA [54, 48], tightened layer-wise convex relaxations [59] and barrier-oriented analyses (e.g., for simplex inputs or active-set scaling) show how to systematically overcome single-neuron limits. Beyond pure LP, SDP-based relaxations with linear cuts capture neuron coupling when affordable [8, 40, 44, 18]. Orthogonal to envelopes, NN-specific cutting planes have been injected directly into bound propagation e.g., general cutting planes (GCP-CROWN [76]) and conflict-driven cuts (BICCOS [79]) to strengthen node-wise relaxations before or alongside BaB. Exact formulations (MILP/SMT) and their strengthened variants remain an important baseline and hybrid component [60, 2, 35, 36], though scalability constraints often shift emphasis toward stronger relaxations and BaB. Recent work also tightens bounds via optimization-driven rolling-horizon decomposition, effectively performing layerwise optimization-based bound tightening to improve intermediate bounds [78]. Our *Complete Clipping* is orthogonal: instead of inventing new global envelopes, we reuse **existing** linear constraints produced during verification to tighten per-neuron affine bounds inside the same bound-propagation framework. On the other hand, Contract-Simple [7] contracts a zonotope using one output inequality. We generalize this in a way tailored to bound propagation: *Relaxed Clipping* applies *arbitrary* linear constraints from final-layer bounds or activation branchings to a box proxy with closed-form updates, avoiding repeated LP solves while yielding globally useful domain reductions reused across all neurons.

6 Conclusion

We proposed **Clip-and-Verify**, a framework that enhances the Branch-and-Bound paradigm in neural network verification through our novel *linear constraint-driven domain clipping* algorithms. An efficient and light-weight paradigm for tightening intermediate-layer bounds has been a remaining gap in neural network verification that has finally been addressed by our framework. Limitation and broader impacts discussed in Appendix E.

Acknowledgements

Huan Zhang is supported in part by the AI2050 program at Schmidt Sciences (AI2050 Early Career Fellowship) and NSF (IIS-2331967). Grani A. Hanasusanto is supported in part by NSF (CCF-2343869 and ECCS-2404413). We thank the anonymous reviewers for their constructive feedback. In particular, we acknowledge Reviewer vsjp for their invaluable insights on the theoretical foundations of our work, particularly for identifying the connection between our dual optimization approach and the continuous knapsack problem.

References

- [1] Matthias Althoff. An introduction to CORA 2015. In *Proc. of the 1st and 2nd Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 120–151. EasyChair, December 2015.
- [2] Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 183(1-2):3–39, 2020.
- [3] David Applegate, Mateo Díaz, Oliver Hinder, Haihao Lu, Miles Lubin, Brendan O’Donoghue, and Warren Schudy. Practical large-scale linear programming using primal-dual hybrid gradient. *Advances in Neural Information Processing Systems*, 34:20243–20257, 2021.
- [4] David Applegate, Mateo Díaz, Oliver Hinder, Haihao Lu, Miles Lubin, Brendan O’Donoghue, and Warren Schudy. Pdlp: A practical first-order method for large-scale linear programming. *arXiv preprint arXiv:2501.07018*, 2025.
- [5] Stanley Bak. nnenum: Verification of relu neural networks with optimized abstraction refinement. In *NASA Formal Methods Symposium*, pages 19–36. Springer, 2021.
- [6] Stanley Bak, Changliu Liu, and Taylor Johnson. The second international verification of neural networks competition (vnn-comp 2021): Summary and results. *arXiv preprint arXiv:2109.00498*, 2021.
- [7] Stanley Bak, Hoang-Dung Tran, Kerianne Hobbs, and Taylor T. Johnson. Improved geometric path enumeration for verifying relu neural networks. In *Proceedings of the 32nd International Conference on Computer Aided Verification*. Springer, 2020.
- [8] Ben Batten, Panagiotis Kouvaros, Alessio Lomuscio, and Yang Zheng. Efficient neural network verification via layer-based semidefinite relaxations and linear cuts. In *IJCAI*, pages 2184–2190, 2021.
- [9] Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. Efficient verification of relu-based neural networks via dependency analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3291–3299, 2020.
- [10] 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.
- [11] Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T Johnson. The fourth international verification of neural networks competition (vnn-comp 2023): Summary and results. *arXiv preprint arXiv:2312.16760*, 2023.
- [12] Robin Brown, Edward Schmerling, Navid Azizan, and Marco Pavone. A unified view of sdp-based neural network verification through completely positive programming, 2022.
- [13] 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.
- [14] Rudy R Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda. A unified view of piecewise linear neural network verification. *Advances in Neural Information Processing Systems*, 31, 2018.
- [15] Nicholas Carlini, Guy Katz, Clark W. Barrett, and David L. Dill. Ground-truth adversarial examples. *CoRR*, abs/1709.10207, 2017.
- [16] Shaoru Chen, Lekan Molu, and Mahyar Fazlyab. Verification-aided learning of neural network barrier functions with termination guarantees. *arXiv preprint arXiv:2403.07308*, 2024.
- [17] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. *CoRR*, abs/1705.01040, 2017.

- [18] Hong-Ming Chiu, Hao Chen, Huan Zhang, and Richard Zhang. Sdp-crown: Efficient bound propagation for neural network verification with tightness of semidefinite programming. In *International Conference on Machine Learning (ICML)*, 2025. ICML 2025.
- [19] Sumanth Dathathri, Krishnamurthy Dvijotham, Alexey Kurakin, Aditi Raghunathan, Jonathan Uesato, Rudy R Bunel, Shreya Shankar, Jacob Steinhardt, Ian Goodfellow, Percy S Liang, et al. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
- [20] Alessandro De Palma, Harkirat Singh Behl, Rudy Bunel, Philip H. S. Torr, and M. Pawan Kumar. Scaling the convex barrier with active sets. *International Conference on Learning Representations (ICLR)*, 2021.
- [21] 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.
- [22] Hai Duong, Linhan Li, ThanhVu Nguyen, and Matthew Dwyer. A dpll (t) framework for verifying deep neural networks. *arXiv preprint arXiv:2307.10266*, 2023.
- [23] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*, 2018.
- [24] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.
- [25] Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15*, pages 269–286. Springer, 2017.
- [26] Mahyar Fazlyab, Manfred Morari, and George J Pappas. Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. *IEEE Transactions on Automatic Control*, 2020.
- [27] Claudio Ferrari, Mark Niklas Muller, Nikola Jovanovic, and Martin Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. *arXiv preprint arXiv:2205.00263*, 2022.
- [28] Matteo Fischetti and Jason Jo. Deep neural networks as 0-1 mixed integer linear programs: A feasibility study. *CoRR*, abs/1712.06174, 2017.
- [29] Timon Gehr, Matthew Mirman, Dana Drachler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018.
- [30] Julien Girard-Satabin, Michele Alberti, Francois Bobot, Zakaria Chihani, and Augustin Lemesle. Caesar: A platform for characterizing artificial intelligence safety and robustness. *arXiv preprint arXiv:2206.03044*, 2022.
- [31] Vahid Hashemi, Panagiotis Kouvaros, and Alessio Lomuscio. Osip: Tightened bound propagation for the verification of relu neural networks. In *Software Engineering and Formal Methods: 19th International Conference, SEFM 2021, Virtual Event, December 6–10, 2021, Proceedings*, page 463–480, Berlin, Heidelberg, 2021. Springer-Verlag.
- [32] Patrick Henriksen and Alessio Lomuscio. Efficient neural network verification via adaptive refinement and adversarial search. In *ECAI 2020*, pages 2513–2520. IOS Press, 2020.
- [33] Patrick Henriksen and Alessio Lomuscio. Deepsplit: An efficient splitting method for neural network verification via indirect effect analysis. In *Proceedings of the 30th international joint conference on artificial intelligence (IJCAI21)*, pages 2549–2555, 2021.
- [34] IBM. Ibm ilog cplex optimizer, 2022.

- [35] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I* 30, pages 97–117. Springer, 2017.
- [36] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification (CAV)*, 2019.
- [37] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer Berlin, Heidelberg, 2004.
- [38] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.
- [39] Panagiotis Kouvaros and Alessio Lomuscio. Towards scalable complete verification of relu neural networks via dependency-based branching. In *IJCAI*, pages 2643–2650, 2021.
- [40] Jianglin Lan, Yang Zheng, and Alessio Lomuscio. Tight neural network verification via semidefinite relaxations and linear reformulations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7272–7280, 2022.
- [41] Haoyu Li, Xiangru Zhong, Bin Hu, and Huan Zhang. Neural contraction metrics with formal guarantees for discrete-time nonlinear dynamical systems. In *7th Annual Learning for Dynamics & Control Conference*, pages 1447–1459. PMLR, 2025.
- [42] Haoyu Li, Xiangru Zhong, Bin Hu, and Huan Zhang. Two-stage learning of stabilizing neural controllers via zubov sampling and iterative domain expansion. *arXiv preprint arXiv:2506.01356*, 2025.
- [43] Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks. *CoRR*, abs/1706.07351, 2017.
- [44] Ziye Ma and Somayeh Sojoudi. Strengthened sdp verification of neural network robustness via non-convex cuts. *arXiv preprint arXiv:2010.08603*, pages 715–727, 2020.
- [45] David R. Morrison, Sheldon H. Jacobson, Jason J. Sauppe, and Edward C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016.
- [46] Christoph Müller, Francois Serre, Gagandeep Singh, Markus Püschel, and Martin Vechev. Scaling polyhedral neural network verification on gpus. *Proceedings of Machine Learning and Systems*, 3:733–746, 2021.
- [47] Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. Precise multi-neuron abstractions for neural network certification. *arXiv preprint arXiv:2103.03638*, 2021.
- [48] Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. Prima: general and precise neural network certification via scalable convex hull approximations. *Proceedings of the ACM on Programming Languages*, 6(POPL):1–33, 2022.
- [49] Mark Niklas Müller, Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T. Johnson. The third international verification of neural networks competition (vnn-comp 2022): Summary and results, 2023.
- [50] Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.
- [51] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

- [52] Karsten Scheibler, Leonore Winterer, Ralf Wimmer, and Bernd Becker. Towards verification of artificial neural networks. In *MBMV*, pages 30–40, 2015.
- [53] Zhouxing Shi, Qirui Jin, Zico Kolter, Suman Jana, Cho-Jui Hsieh, and Huan Zhang. Neural network verification with branch-and-bound for general nonlinearities. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2025.
- [54] Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. Beyond the single neuron convex barrier for neural network certification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.
- [55] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.
- [56] 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.
- [57] Harkirat Singh, M Pawan Kumar, Philip Torr, and Krishnamurthy Dj Dvijotham. Overcoming the convex barrier for simplex inputs. In *Advances in Neural Information Processing Systems*, 2021.
- [58] Chuangchuang Sun, Dong-Ki Kim, and Jonathan P How. Romax: Certifiably robust deep multiagent reinforcement learning via convex relaxation. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 5503–5510. IEEE, 2022.
- [59] Christian Tjandraatmadja, Ross Anderson, Joey Huchette, Will Ma, Krunal Patel, and Juan Pablo Vielma. The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
- [60] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.
- [61] Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.
- [62] Hoang-Dung Tran, Xiaodong Yang, Diego Manzananas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T Johnson. Nnv: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification*, pages 3–17. Springer, 2020.
- [63] Andreas Venzke and Spyros Chatzivasileiadis. Verification of neural network behaviour: Formal guarantees for power system applications. *IEEE Transactions on Smart Grid*, 12(1):383–397, 2020.
- [64] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.
- [65] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *USENIX Security Symposium*, 2018.
- [66] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in Neural Information Processing Systems*, 34:29909–29921, 2021.
- [67] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. Towards fast computation of certified robustness for relu networks. In *International Conference on Machine Learning*, pages 5273–5282, 2018.

- [68] Eric Wong and J Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML*, 2018.
- [69] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning (ICML)*, 2018.
- [70] Eric Wong, Tim Schneider, Joerg Schmitt, Frank R Schmidt, and J Zico Kolter. Neural network virtual sensors for fuel injection quantities with provable performance specifications. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1753–1758. IEEE, 2020.
- [71] 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.
- [72] Junlin Wu, Andrew Clark, Yiannis Kantaros, and Yevgeniy Vorobeychik. Neural lyapunov control for discrete-time systems. *Advances in Neural Information Processing Systems*, 36:2939–2955, 2023.
- [73] Kai Yuanqing Xiao, Vincent Tjeng, Nur Muhammad (Mahi) Shafiullah, and Aleksander Madry. Training for faster adversarial robustness verification via inducing relu stability. *CoRR*, abs/1809.03008, 2018.
- [74] 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. *International Conference on Learning Representations (ICLR)*, 2021.
- [75] Lujie Yang, Hongkai Dai, Zhouxing Shi, Cho-Jui Hsieh, Russ Tedrake, and Huan Zhang. Lyapunov-stable neural control for state and output feedback: a novel formulation. In *Proceedings of the 41st International Conference on Machine Learning, ICML’24*. JMLR.org, 2024.
- [76] 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. *Advances in Neural Information Processing Systems*, 35:1656–1670, 2022.
- [77] 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.
- [78] Haoruo Zhao, Hassan Hijazi, Haydn Jones, Juston Moore, Mathieu Tanneau, and Pascal Van Hentenryck. Bound tightening using rolling-horizon decomposition for neural network verification. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 21st International Conference, CPAIOR 2024, Uppsala, Sweden, May 28–31, 2024, Proceedings, Part II*, page 289–303, Berlin, Heidelberg, 2024. Springer-Verlag.
- [79] Duo Zhou, Christopher Brix, Grani A Hanasusanto, and Huan Zhang. Scalable neural network verification with branch-and-bound inferred cutting planes. *Advances in Neural Information Processing Systems*, 37:29324–29353, 2024.

Appendix

Contents

A	Formulations	17
A.1	Unstable Neurons and Complexity	17
A.2	MIP Formulation and LP & Planet Relaxation	18
A.3	Formulating Bounding Hyperplanes in Linear Bound Propagation	19
A.4	Verification Properties Given as Boolean Logical Formulae	20
B	Proofs	21
B.1	Statement and proof of Lemma B.1	21
B.2	Proof of Theorem 3.1	21
B.3	Equivalence to the Continuous Knapsack Problem	23
B.4	Proof of Theorem 3.2	23
B.5	Proof of Order Dependency	25
C	Algorithms	25
C.1	Overview	25
C.2	Sequential Clipping for Multiple Constraints	26
C.3	Algorithms for Clip-and-Verify in Input Branch-and-bound Scheme	27
C.4	Algorithms for Clip-and-Verify in Activation Branch-and-bound Scheme	28
C.5	2D Toy Example	29
D	Experiments	33
D.1	Experiments Settings	33
D.2	Ablation Studies	34
D.3	Detailed Comparison with LP Solvers	35
D.4	Details of Heuristics	36
E	Limitation and Broader Impacts	38

A Formulations

A.1 Unstable Neurons and Complexity

Unstable neurons for General Activations Let $z_j^{(i)}$ be the pre-activation of neuron (i, j) in layer i number j with interval bounds $[\underline{z}_j^{(i)}, \bar{z}_j^{(i)}]$, and let the component-wise activation be $\phi^{(i)}(\cdot)$. Given a sound linear relaxation (convex envelope) for the scalar graph of $\phi^{(i)}$ over $[\underline{z}_j^{(i)}, \bar{z}_j^{(i)}]$, denote by $l_j^{(i)}(z)$ and $u_j^{(i)}(z)$ its lower and upper affine bounds, and define the envelope gap (or relaxation error):

$$\delta_j^{(i)} \triangleq \max_{z \in [\underline{z}_j^{(i)}, \bar{z}_j^{(i)}]} (u_j^{(i)}(z) - l_j^{(i)}(z)). \quad (8)$$

We call neuron (i, j) *stable* if $\delta_j^{(i)} = 0$ (the relaxation is exact on the interval), and *unstable* if $\delta_j^{(i)} > 0$. This definition is activation-agnostic: for piecewise-linear (PWL) activations, stability is equivalent to $[\underline{z}_j^{(i)}, \bar{z}_j^{(i)}]$ lying inside a single linear piece (e.g., for ReLU it reduces to $\bar{z}_j^{(i)} \leq 0$ or $\underline{z}_j^{(i)} \geq 0$); for smooth nonlinear activations one typically adopts a PWL relaxation, in which case instability means the interval spans at least two segments (hence a nonzero envelope gap).

Complexity. Let U be the number of unstable neurons at a BaB node. Once every unstable neuron has its activation region (piece) fixed, the network reduces to an affine map over the input box, and the remaining subproblem is convex. For PWL activations where each unstable neuron has at most

K admissible pieces,¹ the number of global activation patterns is bounded by K^U . Consequently, any exact verifier that distinguishes activation pieces (e.g., MILP/SMT encodings with one discrete choice per unstable neuron, or BaB that branches on activation pieces) faces, in the worst case,

$$\# \text{ subproblems} \in \Omega(K^U) \quad \text{and hence} \quad \text{time} = \Omega(T_{\text{bound}} \cdot K^U),$$

where T_{bound} is the per-node cost to evaluate/tighten bounds. In particular, for ReLU networks $K = 2$, yielding the familiar exponential scaling $\Omega(2^U)$; thus any mechanism that reduces U (e.g., by stabilizing neurons via tighter pre-activation intervals) produces a multiplicative reduction in worst-case search, by a factor of $K^{\Delta U}$ when U decreases by ΔU .

A.2 MIP Formulation and LP & Planet Relaxation

Early approaches pursue exact guarantees for verification through Mixed-Integer Linear Programming (MILP) [17, 43, 23, 28, 61, 73] and Satisfiability Modulo Theories (SMT) [52, 35, 15, 25].

The MIP Formulation The mixed integer programming (MIP) formulation is the root of many NN verification algorithms. Given the RELU activation function’s piecewise linearity, the model requires binary encoding variables, or ReLU indicators δ only for unstable neurons. We formulate the optimization problem aiming to minimize the function $f(\mathbf{x})$, subject to a set of constraints that encapsulate the DNN’s architecture and the perturbation limits around a given input $\mathbf{x}$, as follows:

$$f^* = \min_{\mathbf{x}, \hat{\mathbf{z}}, \delta} f(\mathbf{x}) \quad \text{s.t. } f(\mathbf{x}) = z^{(L)}; \hat{\mathbf{z}}^{(0)} = \mathbf{x} \in \mathcal{X} \quad (9a)$$

$$\hat{\mathbf{z}}^{(i)} = \mathbf{W}^{(i)} \hat{\mathbf{z}}^{(i-1)} + \mathbf{b}^{(i)}; \quad i \in [L] \quad (9b)$$

$$\mathcal{I}^{+(i)} := \{j : l_j^{(i)} \geq 0\} \quad (9c)$$

$$\mathcal{I}^{-(i)} := \{j : u_j^{(i)} \leq 0\} \quad (9d)$$

$$\mathcal{I}^{(i)} := \{j : l_j^{(i)} < 0, u_j^{(i)} > 0\} \quad (9e)$$

$$\mathcal{I}^{+(i)} \cup \mathcal{I}^{-(i)} \cup \mathcal{I}^{(i)} = \mathcal{J}^i \quad (9f)$$

$$\hat{x}_j^{(i)} \geq 0; j \in \mathcal{I}^{(i)}, i \in [L-1] \quad (9g)$$

$$\hat{z}_j^{(i)} \geq z_j^{(i)}; j \in \mathcal{I}^{(i)}, i \in [L-1] \quad (9h)$$

$$\hat{z}_j^{(j)} \leq u_j^{(i)} \delta_j^{(i)}; j \in \mathcal{I}^{(i)}, i \in [L-1] \quad (9i)$$

$$\hat{z}_j^{(i)} \leq z_j^{(i)} - l_j^{(i)} (1 - \delta_j^{(i)}); j \in \mathcal{I}^{(i)}, i \in [L-1] \quad (9j)$$

$$\delta_j^{(i)} \in \{0, 1\}; j \in \mathcal{I}^{(i)}, i \in [L-1] \quad (9k)$$

$$\hat{z}_j^{(i)} = z_j^{(i)}; j \in \mathcal{I}^{+(i)}, i \in [L-1] \quad (9l)$$

$$\hat{z}_j^{(i)} = 0; j \in \mathcal{I}^{-(i)}, i \in [L-1] \quad (9m)$$

To initialize intermediate bounds for each neuron, we replace the original objective $f(\mathbf{x})$ with the neuron’s pre-activation value $z_j^{(i)}$. This lets us solve the following bounds for every neuron j in layer i , with $i \in [L-1]$ and $j \in \mathcal{J}^{(i)}$:

$$l_j^{(i)} = \min_{\mathbf{x} \in \mathcal{X}} f_j^{(i)}(\mathbf{x}), \quad u_j^{(i)} = \max_{\mathbf{x} \in \mathcal{X}} f_j^{(i)}(\mathbf{x}).$$

Here, the set $\mathcal{J}^{(i)}$ comprises all neurons in layer i , which can be categorized into three groups: ‘active’ ($\mathcal{I}^{+(i)}$), ‘inactive’ ($\mathcal{I}^{-(i)}$), and ‘unstable’ ($\mathcal{I}^{(i)}$).

Next, the MIP formulation is initialized with the constraints

$$l_j^{(i)} \leq z_j^{(i)} \leq u_j^{(i)},$$

¹For ReLU, $K = 2$ (inactive/active). For HardTanh or ReLU6, $K \leq 3$. For smooth activations under an S -segment relaxation, take $K = S$.

across all neurons and layers i . These bounds can be computed recursively, propagating from the first layer up to the i -th layer. However, since MIP problems involve integer variables, they are generally NP-hard, reflecting the computational challenge of this approach.

The LP and Planet relaxation. By relaxing the binary variables in (9k) to $\delta_j^{(i)} \in [0, 1]$, $j \in \mathcal{I}^{(i)}$, $i \in [L - 1]$, we can get the LP relaxation formulation. By replacing the constraints in (9i), (9j), (9k) with

$$\hat{z}_j^{(i)} \leq \frac{u_j^{(i)}}{u_j^{(i)} - l_j^{(i)}} (z_j^{(i)} - l_j^{(i)}); \quad j \in \mathcal{I}^{(i)}, i \in [L - 1] \quad (10)$$

we can eliminate the δ variables and get the well-known Planet relaxation formulation [25]. Both of these two relaxations are solvable in polynomial time to yield lower bounds.

A.3 Formulating Bounding Hyperplanes in Linear Bound Propagation

In a feedforward network, $\underline{\mathbf{A}}^{(i)}$, $\overline{\mathbf{A}}^{(i)}$, $\underline{\mathbf{c}}^{(i)}$ and $\overline{\mathbf{c}}^{(i)}$ must be derived for every linear layer preceding an activation layer, as well as the final layer of the network. In order to derive the hyperplane coefficients ($\underline{\mathbf{A}}^{(i)}/\overline{\mathbf{A}}^{(i)}$) and biases ($\underline{\mathbf{c}}^{(i)}/\overline{\mathbf{c}}^{(i)}$), at this i th layer, all preceding activation layers must have already had their inputs bounded. The following lemma describes how a ReLU activation layer may be relaxed which will be useful for defining bounding hyperplanes, $\underline{\mathbf{A}}^{(i)}$, $\overline{\mathbf{A}}^{(i)}$, $\underline{\mathbf{c}}^{(i)}$ and $\overline{\mathbf{c}}^{(i)}$.

Lemma A.1. (Relaxation of a ReLU layer in CROWN). Given the lower and upper bounds of $\mathbf{z}_j^{(i-1)}$, denoted as $l_j^{(i-1)}$ and $u_j^{(i-1)}$, respectively, the linear layer proceeding the ReLU activation layer may be lower-bounded element-wise by the following inequality:

$$\mathbf{z}^{(i)} = \mathbf{W}^{(i)} \sigma(\mathbf{z}^{(i-1)}) \geq \mathbf{W}^{(i)} \mathbf{D}^{(i-1)} \mathbf{z}^{(i-1)} + \mathbf{W}^{(i)} \underline{\mathbf{b}}^{(i-1)} \quad (11)$$

where $\mathbf{D}^{(i-1)}$ is a diagonal matrix with shape $\mathbb{R}^{n_{i-1} \times n_{i-1}}$ whose off-diagonal entries are 0, and on-diagonal entries are defined as:

$$D_{j,j}^{(i-1)} := \begin{cases} 1, & l_j^{(i-1)} \geq 0 \\ 0, & u_j^{(i-1)} \leq 0 \\ \alpha_j^{(i-1)}, & l_j^{(i-1)} < 0 < u_j^{(i-1)} \text{ and } W_j^{(i)} \geq 0 \\ \frac{u_j^{(i-1)}}{u_j^{(i-1)} - l_j^{(i-1)}}, & l_j^{(i-1)} < 0 < u_j^{(i-1)} \text{ and } W_j^{(i)} < 0 \end{cases} \quad (12)$$

and $\underline{\mathbf{b}}_j^{(i-1)}$ is a vector with shape $\mathbb{R}^{n_{i-1}}$ whose elements are defined as:

$$\underline{\mathbf{b}}_j^{(i-1)} := \begin{cases} 0, & l_j^{(i-1)} > 0 \text{ or } u_j^{(i-1)} \leq 0 \\ 0, & l_j^{(i-1)} < 0 < u_j^{(i-1)} \text{ and } W_j^{(i)} \geq 0 \\ -\frac{u_j^{(i-1)} l_j^{(i-1)}}{u_j^{(i-1)} - l_j^{(i-1)}}, & l_j^{(i-1)} < 0 < u_j^{(i-1)} \text{ and } W_j^{(i)} < 0 \end{cases} \quad (13)$$

In the above definitions, $\alpha_j^{(i-1)}$ is a parameter in range $[0, 1]$ and may be fixed or optimized as in [74].

Proof. For the j^{th} ReLU at the $(i - 1)^{\text{th}}$ layer, it's result may be bounded as follows:

$$\alpha_j^{(i-1)} z_j^{(i-1)} \leq \sigma(z_j^{(i-1)}) \leq \frac{u_j^{(i-1)}}{u_j^{(i-1)} - l_j^{(i-1)}} (z_j^{(i-1)} - l_j^{(i-1)}). \quad (14)$$

The right-hand side holds as this is the Planet-relaxation defined by equation (10). For the left-hand side, we first consider when $z_j^{(i-1)} \leq 0$. For every input in this range, the result of the ReLU is $\sigma(z_j^{(i-1)}) = 0$. $\alpha_j^{(i-1)} z_j^{(i-1)}$ forms a line for which inputs in this range will always produce a

non-positive result when $\alpha_j^{(i-1)} \in [0, 1]$. For inputs in the range $z_j^{(i-1)} \geq 0$, the result of the ReLU is $\sigma(z_j^{(i-1)}) = z_j^{(i-1)}$. This result is never exceeded by $\alpha_j^{(i-1)} z_j^{(i-1)}$ when $\alpha_j^{(i-1)} \in [0, 1]$.

When the result, $\sigma(z_j^{(i-1)})$, is multiplied by a scalar such as $W_j^{(i)}$, a valid lower-bound of $W_j^{(i)} \sigma(z_j^{(i-1)})$ requires a lower bound on $\sigma(z_j^{(i-1)})$ when $W_j^{(i)} \geq 0$, and an upper bound on $\sigma(z_j^{(i-1)})$ when $W_j^{(i)} < 0$. Such lower and upper bounds are indeed produced by $D_{j,j}^{(i-1)}$ and $b_j^{(i-1)}$, whose definitions are derived from the inequality displayed in equation (14). This concludes the proof. $\square$

Lemma A.1 suggests a recursive approach to bounding a neural network as the bounds at the i^{th} layer depends on the bounds of the layer preceding it due to the dependence on $l_j^{(i-1)}$ and $u_j^{(i-1)}$. This is indeed the case, and we may define our hyperplane coefficients as $\underline{A}^{(i)} = \Omega^{(i,1)} W^{(1)}$ where

$$\Omega^{(i,k)} := \begin{cases} W^{(i)} D^{(i-1)} \Omega^{(i-1)}, & i > k \\ I, & i = k \end{cases} \quad (15)$$

To collect the remaining terms, we set $\underline{c}^{(i)} = \sum_{k=2}^i (\Omega^{(i,k)} W^{(k)} \underline{b}^{(k-1)}) + \sum_{k=1}^i (\Omega^{(i,k)} b^{(k)})$. To obtain an upper bound, Lemma A.1 and its proof may be adjusted accordingly where appearances of the inequalities $W^{(i)} \geq 0$ and $W^{(i)} < 0$ are flipped. In doing so, we may repeat this recursive process in order to obtain $\overline{A}^{(i)}$ and $\overline{c}^{(i)}$.

Though we have described how a ReLU feedforward network may be bounded, appropriately updating the definitions of $D^{(i)}$ and $b^{(i)}$ allows feedforward networks with general activation functions (that act element-wise) to be bounded. Such a general formulation is described in [77] that is similar to the template described above, and goes into further detail on how this formulation may be extended to *quadratic* bound propagation.

A.4 Verification Properties Given as Boolean Logical Formulae

$$f(x) \geq 0, \quad \forall x \in \mathcal{X} \quad (16)$$

Equation (16) is referred to as the *canonical* formulation [14, 13] as it is general enough to encompass any property involving boolean logical formulas over linear inequalities. To make this idea clear, consider a neural network whose output dimension is $n_L = 3$. We may wish to verify that the output corresponding to the true label, y_0 , is always greater than all other outputs, y_1 and y_2 , for every input in $\mathcal{X}$. Explicitly, we want to verify:

$$(y_0 > y_1) \wedge (y_0 > y_2), \quad \forall x \in \mathcal{X}. \quad (17)$$

To represent this formula, we may first introduce a new linear layer whose weight matrix is defined as C , sometimes referred to as a *specification matrix*:

$$C := \begin{bmatrix} 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix}. \quad (18)$$

This specification matrix allows the output neurons to be compared against one another. Applying this matrix to the output vector of the network, denoted as $y := [y_0, y_1, y_2]^T$, results in the following new output:

$$\begin{bmatrix} y_0 - y_1 \\ y_0 - y_2 \end{bmatrix} = Cy. \quad (19)$$

Next, we append a *MinPool* layer to the network which returns the minimum of these two results. Now, we may define a new neural network, $f'(x)$, which appends the aforementioned *MinPool* and linear layer to $f(x)$, resulting in the final output, $\min\{y_0 - y_1, y_0 - y_2\}$. It should now be apparent that if the property described by equation (17) can be verified as true, then the following must hold:

$$\min_{x \in \mathcal{X}} f'(x) > 0. \quad (20)$$

We may now proceed to produce a lower bound, $f'(x)$, and only until $f'(x) > 0$, can we formally state that the network will correctly classify all inputs in $\mathcal{X}$. In the scenario that we instead want to verify the property $(y_0 > y_1 \vee y_0 > y_2)$ for all inputs in $\mathcal{X}$, we may simply redefine $f'(x)$ by replacing the *MinPool* layer with a *MaxPool* layer which would return $\max\{y_0 - y_1, y_0 - y_2\}$.

Finally, suppose we want to verify that the output corresponding to the true label is greater than all other outputs with some additional margin m such that $(y_0 > y_1 + m \wedge y_0 > y_2 + m), \forall x \in \mathcal{X}$ where $m > 0$ (i.e. the network should always classify the first label with additional relative confidence defined by m). Then the only modification is to also incorporate a bias vector, $t := [-m, -m]^\top$, at the final linear layer where C was defined. This bias vector is sometimes referred to as the *threshold*. This results in the final output, $f'(x) = \min\{y_0 - y_1 - m, y_0 - y_2 - m\}$. Hence, equation (16) is general enough to encompass more complex queries.

B Proofs

B.1 Statement and proof of Lemma B.1

We first introduce a preparatory result that establishes a closed-form solution for a linear optimization problem over a hyper-rectangular feasible set.

Lemma B.1 (Dual Norm Concretization for Hyper-Rectangular Domains). *Let $\epsilon \in \mathbb{R}^n$ with $\epsilon_i > 0$ for all $i \in [n]$, and define the hyper-rectangular domain:*

$$\mathcal{X} = \{x \in \mathbb{R}^n : \hat{x} - \epsilon \leq x \leq \hat{x} + \epsilon\}.$$

For any vector $v \in \mathbb{R}^n$, the following hold:

$$\begin{aligned} \max_{x \in \mathcal{X}} v^\top x &= v^\top \hat{x} + |v|^\top \epsilon, \\ \min_{x \in \mathcal{X}} v^\top x &= v^\top \hat{x} - |v|^\top \epsilon, \end{aligned}$$

where $|v| \in \mathbb{R}_+^n$ denotes the component-wise absolute value of v .

Proof. We begin by rewriting the feasible set as

$$\mathcal{X} = \{x \in \mathbb{R}^n : \hat{x} - \epsilon \leq x \leq \hat{x} + \epsilon\} = \{\hat{x} + \epsilon \circ x : x \in \mathbb{R}^n, \|x\|_\infty \leq 1\},$$

where $\circ$ denotes the Hadamard (component-wise) product. Then, the maximization problem becomes

$$\begin{aligned} \max_{x \in \mathcal{X}} v^\top x &= \max_{\|x\|_\infty \leq 1} v^\top \hat{x} + (v \circ \epsilon)^\top x \\ &= v^\top \hat{x} + \sum_{i=1}^n |v_i| \epsilon_i = v^\top \hat{x} + |v|^\top \epsilon, \end{aligned}$$

where the final equality follows from the definition of the dual norm of the ∞ -norm. The derivation for the minimization problem follows analogously by replacing the maximization with a minimization, which flips the sign of $|v|^\top \epsilon$. Thus, the claim follows. $\square$

B.2 Proof of Theorem 3.1

Proof. The problem (2) is a linear program over a compact convex set $\mathcal{X}$ with an additional linear constraint. Let feasible set $\mathcal{F} = \{x \in \mathcal{X} : g^\top x + h \leq 0\}$ be the feasible set for (2).

Case 1: $\mathcal{F} = \emptyset$, then by definition, $L^* = +\infty$. In this case, it implies that for all $x \in \mathcal{X}$, $g^\top x + h > 0$.

Case 2: $\mathcal{F}$ is non-empty. Since $\mathcal{X}$ is compact and $\mathcal{F}$ is a closed non-empty subset of $\mathcal{X}$ (as it's the intersection of $\mathcal{X}$ with a closed half-space), $\mathcal{F}$ is also compact. The objective function $a^\top x + c$ is continuous. Therefore, L^* is finite and attained. We introduce the Lagrangian for problem (2) by partially dualizing the constraint $g^\top x + h \leq 0$:

$$\mathcal{L}(x, \beta) = (a^\top x + c) + \beta(g^\top x + h) \quad \text{for } x \in \mathcal{X}, \beta \geq 0.$$

The primal problem can be written as:

$$L^* = \min_{\mathbf{x} \in \mathcal{X}} \sup_{\beta \geq 0} \mathcal{L}(\mathbf{x}, \beta)$$

To swap the min and sup, we can apply Sion's Minimax Theorem. Let $K = \mathcal{X}$ (a compact convex set in $\mathbb{R}^n$) and $M = \{\beta \in \mathbb{R} : \beta \geq 0\}$ (a closed convex set in $\mathbb{R}$). The function $\mathcal{L}(\mathbf{x}, \beta)$ has the following properties: 1) For any fixed $\beta \in M$, $\mathcal{L}(\mathbf{x}, \beta) = (\mathbf{a} + \beta \mathbf{g})^\top \mathbf{x} + (c + \beta h)$ is linear in $\mathbf{x}$, and thus convex and continuous on K . 2) For any fixed $\mathbf{x} \in K$, $\mathcal{L}(\mathbf{x}, \beta) = (\mathbf{g}^\top \mathbf{x} + h)\beta + (\mathbf{a}^\top \mathbf{x} + c)$ is linear in β , and thus concave and continuous on M . Since these conditions are met, Sion's Minimax Theorem states:

$$\min_{\mathbf{x} \in K} \sup_{\beta \in M} \mathcal{L}(\mathbf{x}, \beta) = \sup_{\beta \in M} \min_{\mathbf{x} \in K} \mathcal{L}(\mathbf{x}, \beta)$$

Therefore,

$$L^* = \sup_{\beta \geq 0} \min_{\mathbf{x} \in \mathcal{X}} \mathcal{L}(\mathbf{x}, \beta)$$

Since $\mathcal{F}$ is non-empty, L^* is finite, implying that the supremum is attained (or is the limit if approached at infinity, but $D(\beta)$ is continuous), so we can write max instead of sup. Let $d(\beta) = \min_{\mathbf{x} \in \mathcal{X}} \mathcal{L}(\mathbf{x}, \beta)$.

$$\begin{aligned} d(\beta) &= \min_{\mathbf{x} \in \mathcal{X}} (\mathbf{a}^\top \mathbf{x} + c + \beta \mathbf{g}^\top \mathbf{x} + \beta h) \\ &= \min_{\mathbf{x} \in \mathcal{X}} ((\mathbf{a} + \beta \mathbf{g})^\top \mathbf{x}) + c + \beta h \end{aligned}$$

The inner minimization is optimizing a linear function $(\mathbf{a} + \beta \mathbf{g})^\top \mathbf{x}$ over the hyper-rectangle $\mathcal{X}$. Using Lemma B.1 (with $\mathbf{v} = \mathbf{a} + \beta \mathbf{g}$):

$$\min_{\mathbf{x} \in \mathcal{X}} ((\mathbf{a} + \beta \mathbf{g})^\top \mathbf{x}) = (\mathbf{a} + \beta \mathbf{g})^\top \hat{\mathbf{x}} - |\mathbf{a} + \beta \mathbf{g}|^\top \boldsymbol{\epsilon}$$

where $|\mathbf{a} + \beta \mathbf{g}|^\top \boldsymbol{\epsilon} = \sum_{j=1}^n |a_j + \beta g_j| \epsilon_j$. Substituting this into the expression for $d(\beta)$, we get:

$$d(\beta) = (\mathbf{a} + \beta \mathbf{g})^\top \hat{\mathbf{x}} - |\mathbf{a} + \beta \mathbf{g}|^\top \boldsymbol{\epsilon} + c + \beta h$$

This is exactly the dual objective function $D(\beta)$ defined in the theorem statement (3). Thus, we have established that $L^* = \max_{\beta \geq 0} D(\beta)$. Then we analyze the properties of the dual objective $D(\beta)$:

1. Concavity: The dual function $d(\beta)$ is always concave, as it is the pointwise minimum of a family of functions that are affine in β (indexed by $\mathbf{x} \in \mathcal{X}$).
2. Piecewise-Linearity: The term $-|\mathbf{a} + \beta \mathbf{g}|^\top \boldsymbol{\epsilon} = -\sum_{j=1}^n |a_j + \beta g_j| \epsilon_j$ involves the absolute value function. Each term $-|a_j + \beta g_j| \epsilon_j$ is concave and piecewise-linear, with a breakpoint (a point where the slope changes) at $\beta = -a_j/g_j$ (if $g_j \neq 0$). The other terms in $D(\beta)$ are linear in β . Since $D(\beta)$ is a sum of concave piecewise-linear functions and linear functions, it is itself concave and piecewise-linear. The breakpoints of $D(\beta)$ are the collection of all values $\beta = -a_j/g_j \geq 0$ where $g_j \neq 0$.

Thus, we need to maximize the concave, piecewise-linear function $D(\beta)$ over the interval $[0, \infty)$. Since $D(\beta)$ is concave, its maximum over a convex set occurs either at a point where the super-gradient contains zero, or potentially at the boundary point $\beta = 0$. Because $D(\beta)$ is piecewise-linear, its super-gradient $\partial D(\beta)$ is constant within the linear segments between breakpoints. At a breakpoint β_k , the super-gradient is an interval $[\partial D(\beta_k^-), \partial D(\beta_k^+)]$ (the range between the left and right derivatives). The maximum occurs at a point β^* such that $0 \in \partial D(\beta^*)$. This β^* must be either $\beta = 0$ (if the derivative is non-positive for $\beta > 0$) or one of the breakpoints $\beta_k > 0$ where the derivative changes sign from positive to non-positive (i.e., $0 \in [\partial D(\beta_k^-), \partial D(\beta_k^+)]$), or the function increases indefinitely (which corresponds to an infeasible or unbounded primal, but we assumed feasibility and the primal is bounded over the compact $\mathcal{X}$, so L^* is finite, thus the dual maximum is finite).

Therefore, the maximum L^* can be found non-iteratively by:

1. Identifying all non-negative breakpoints $\beta_k = -a_j/g_j \geq 0$.

2. Sorting these unique breakpoints $0 = \beta_0 < \beta_1 < \dots < \beta_p$.
3. Evaluating the derivative (slope) of $D(\beta)$ within each segment (β_k, β_{k+1}) and potentially at $\beta = 0$.
4. Finding the point β^* (either 0 or some β_k) where the slope transitions from non-negative to non-positive. The value $D(\beta^*)$ is the maximum L^* .

This process involves a finite number of analytical calculations (evaluating slopes and function values at breakpoints) rather than iterative optimization, justifying the claim of efficiency. $\square$

B.3 Equivalence to the Continuous Knapsack Problem

The primal problem in (2) is mathematically equivalent to the continuous (or fractional) knapsack problem. We can demonstrate this equivalence through a change of variables. Let $x_0 = \hat{x} - \epsilon$ be the lower bound and $x_1 = \hat{x} + \epsilon$ be the upper bound. We transform $x \in [x_0, x_1]$ to $y \in [0, 1]^n$ using:

$$y_i = \frac{x_i - x_{0,i}}{x_{1,i} - x_{0,i}} = \frac{x_i - (\hat{x}_i - \epsilon_i)}{2\epsilon_i} \implies x_i = x_{0,i} + 2\epsilon_i y_i$$

Substituting this into the primal problem $\min_{x \in \mathcal{X}} \{a^\top x + c \mid g^\top x + h \leq 0\}$ gives:

$$\begin{aligned} \min_{y \in [0,1]^n} \quad & a^\top (x_0 + 2(\epsilon \odot y)) + c \quad \text{s.t.} \quad g^\top (x_0 + 2(\epsilon \odot y)) + h \leq 0 \\ \min_{y \in [0,1]^n} \quad & (2\epsilon \odot a)^\top y + (a^\top x_0 + c) \quad \text{s.t.} \quad (2\epsilon \odot g)^\top y \leq -(g^\top x_0 + h) \end{aligned}$$

Let $r = -2\epsilon \odot a$, $s = 2\epsilon \odot g$, and $t = -(g^\top (\hat{x} - \epsilon) + h)$. The problem becomes the standard knapsack form:

$$\max_{y \in [0,1]^n} \quad r^\top y \quad \text{s.t.} \quad s^\top y \leq t$$

This problem, even with negative coefficients, can be solved with a greedy algorithm [37]. The efficiency ratios r_j/s_j used for sorting in the greedy algorithm are:

$$\frac{r_j}{s_j} = \frac{-2\epsilon_j a_j}{2\epsilon_j g_j} = -a_j/g_j$$

These ratios are identical to the breakpoints in our dual objective function $D(\beta)$. This confirms a line-by-line correspondence: our dual optimization algorithm, which sorts breakpoints to find where the super-gradient contains zero, is algorithmically equivalent to the greedy knapsack algorithm, which sorts by efficiency ratios. Both have the same $\mathcal{O}(n \log n)$ time complexity.

B.4 Proof of Theorem 3.2

Direct Intuitive Proof A more direct and intuitive proof for Theorem 3.2 exists. We wish to solve $\bar{x}_i^{(new)} = \max_{x \in \mathcal{X}} \{x_i \mid a^\top x + c \leq 0\}$. Since only x_i is in the objective, we can set all other variables x_j (for $j \neq i$) to values within their box domain $[\hat{x}_j - \epsilon_j, \hat{x}_j + \epsilon_j]$ that make the constraint $a^\top x + c \leq 0$ as loose as possible, thereby maximizing the “budget” for x_i .

To loosen the constraint, we must minimize the term $\sum_{j \neq i} a_j x_j$. This is achieved by setting each x_j to its extreme:

- If $a_j > 0$, we set x_j to its lower bound, $x_j = \hat{x}_j - \epsilon_j$.
- If $a_j < 0$, we set x_j to its upper bound, $x_j = \hat{x}_j + \epsilon_j$.

This worst-case minimum for the sum can be written compactly as $\sum_{j \neq i} (a_j \hat{x}_j - |a_j| \epsilon_j)$. We substitute this minimum sum back into the constraint:

$$a_i x_i + \sum_{j \neq i} (a_j \hat{x}_j - |a_j| \epsilon_j) + c \leq 0$$

Assuming $a_i > 0$, we can solve for x_i to find its new upper bound:

$$x_i \leq \frac{-\sum_{j \neq i} a_j \hat{x}_j + \sum_{j \neq i} |a_j| \epsilon_j - c}{a_i}$$

This value is $x_i^{(\text{clip})}$, as defined in Theorem 3.2. The final bound $\bar{x}_i^{(\text{new})}$ is the minimum of this value and the original upper bound $\bar{x}_i$. The case for $a_i < 0$ (updating the lower bound $\underline{x}_i$) follows analogously. A detailed proof is shown below:

Proof. We consider the upper bound; the lower bound can be derived analogously. First, we rewrite the input region as

$$\mathcal{X} = \{\mathbf{x} \in \mathbb{R}^n : \underline{\mathbf{x}} \leq \mathbf{x} \leq \bar{\mathbf{x}}\} = \{\mathbf{x} \in \mathbb{R}^n : \hat{\mathbf{x}} - \boldsymbol{\epsilon} \leq \mathbf{x} \leq \hat{\mathbf{x}} + \boldsymbol{\epsilon}\},$$

where $\hat{\mathbf{x}} = \frac{\bar{\mathbf{x}} + \underline{\mathbf{x}}}{2}$ and $\boldsymbol{\epsilon} = \frac{\bar{\mathbf{x}} - \underline{\mathbf{x}}}{2}$. Suppose the linear inequality constraint is given by $\mathbf{a}^\top \mathbf{x} + b \leq 0$, where $\mathbf{a} \in \mathbb{R}^n$ and $b \in \mathbb{R}$. We note that the intersection $\mathcal{X} \cap \{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}^\top \mathbf{x} + b \leq 0\}$ is nonempty if and only if

$$0 \geq \min_{\mathbf{x} \in \mathcal{X}} \mathbf{a}^\top \mathbf{x} + b = \mathbf{a}^\top \hat{\mathbf{x}} + b - \sum_{i=1}^n |a_i| \epsilon_i$$

by Lemma B.1. Henceforth, we will assume this inequality is satisfied.

We now compute

$$\begin{aligned} \bar{x}_i^{(\text{new})} &= \max_{\mathbf{x} \in \mathcal{X}} \{e_i^\top \mathbf{x} : \mathbf{a}^\top \mathbf{x} + b \leq 0\} \\ &= \max_{\mathbf{x} \in \mathcal{X}} \min_{\lambda \in \mathbb{R}_+} e_i^\top \mathbf{x} - \lambda(\mathbf{a}^\top \mathbf{x} + b) \\ &= \min_{\lambda \in \mathbb{R}_+} \max_{\mathbf{x} \in \mathcal{X}} e_i^\top \mathbf{x} - \lambda(\mathbf{a}^\top \mathbf{x} + b) \\ &= \min_{\lambda \in \mathbb{R}_+} e_i^\top \hat{\mathbf{x}} - \lambda(\mathbf{a}^\top \hat{\mathbf{x}} + b) + |e_i - \lambda \mathbf{a}|^\top \boldsymbol{\epsilon}, \end{aligned}$$

where third line follows from Sion's minimax theorem since $\mathcal{X}$ is a compact set, and the final line follows from Lemma B.1.

Rearranging the term yields

$$\bar{x}_i^{(\text{new})} = \min_{\lambda \in \mathbb{R}_+} e_i^\top \hat{\mathbf{x}} + \lambda \left(\sum_{j \neq i} \epsilon_j |a_j| - \mathbf{a}^\top \hat{\mathbf{x}} - b \right) + \epsilon_i |1 - \lambda a_i|. \quad (21)$$

We now analyze three cases for a_i and derive a closed-form expression for the scalar minimization. Recall that we have assumed $0 \geq \mathbf{a}^\top \hat{\mathbf{x}} + c - \sum_{i=1}^n a_i \epsilon_i$ or equivalently $\sum_{j \neq i} \epsilon_j |a_j| - \mathbf{a}^\top \hat{\mathbf{x}} - b \geq -\epsilon_i |a_i|$. We have:

1. $a_i > 0$: The function $\epsilon_i |1 - \lambda a_i|$ attains its minimum at $\lambda = \frac{1}{a_i} > 0$, with slope $\epsilon_i a_i$ on the right and $-\epsilon_i a_i$ on the left. Thus:

- If $|\sum_{j \neq i} \epsilon_j |a_j| - \mathbf{a}^\top \hat{\mathbf{x}} - b| \leq \epsilon_i a_i$ then the minimum of (21) is attained at $\lambda^* = \frac{1}{a_i}$ and

$$\bar{x}_i^{(\text{new})} = \hat{x}_i + \frac{\sum_{j \neq i} \epsilon_j |a_j| - \mathbf{a}^\top \hat{\mathbf{x}} - b}{a_i} = \frac{\sum_{j \neq i} \epsilon_j |a_j| - \sum_{j \neq i} a_j \hat{x}_j - b}{a_i}.$$

- If $\sum_{j \neq i} \epsilon_j |a_j| - \mathbf{a}^\top \hat{\mathbf{x}} - b > \epsilon_i a_i$ then $\lambda^* = 0$ and

$$\bar{x}_i^{(\text{new})} = \hat{x}_i + \epsilon_i = \bar{x}_i.$$

2. $a_i < 0$: In this case, the minimizing $\lambda = \frac{1}{a_i} < 0$ is infeasible to (21). Hence, $\lambda^* = 0$ and $\bar{x}_i^{(\text{new})} = \bar{x}_i$.

3. $a_i = 0$: Here, the objective function in (21) becomes affine in λ . Since $\sum_{j \neq i} \epsilon_j |a_j| - \mathbf{a}^\top \hat{\mathbf{x}} - b \geq 0$, we again have $\lambda^* = 0$ and $\bar{x}_i^{(new)} = \bar{x}_i$.

In summary, if $\epsilon^\top |\mathbf{a}| - \mathbf{a}^\top \hat{\mathbf{x}} - b < 0$ then $\emptyset = \mathcal{X} \cap \{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}^\top \mathbf{x} + c \leq 0\}$; otherwise,

$$\bar{x}_i^{(new)} = \begin{cases} \min \left\{ \frac{\sum_{j \neq i} \epsilon_j |a_j| - \sum_{j \neq i} a_j \hat{x}_j - b}{a_i}, \bar{x}_i \right\} & \text{if } a_i > 0 \\ \bar{x}_i & \text{if } a_i \leq 0. \end{cases}$$

This completes the proof. $\square$

B.5 Proof of Order Dependency

Here we briefly introduce Theorem B.2 to demonstrate why sequential processing the constraints for clipping achieving better results.

Theorem B.2 (Order Dependency of Constraint Intersection). *Let $\mathcal{X} = \{x \in \mathbb{R}^n : x_L \leq x \leq x_U\}$ be a box domain, and let $F_j = \{x \in \mathcal{X} : \mathbf{a}_j^\top x + c_j \leq 0\}$ denote the feasible set for the j -th constraint. Define the full feasible set as $F = \bigcap_{j=1}^m F_j$.*

Let $\pi : \{1, \dots, m\} \rightarrow \{1, \dots, m\}$ be a permutation (i.e., a reordering) of the constraints. For sequential intersection, define:

$$F^{(k)} = \bigcap_{t=1}^k F_{\pi(t)}, \quad k = 1, \dots, m,$$

with $F^{(0)} = \mathcal{X}$.

(Order-Independent Bounds): *The variable-wise bounds satisfy:*

$$\max_{x \in F} x_i \leq \min_j \max_{x \in F_j} x_i \quad \text{and} \quad \min_{x \in F} x_i \geq \max_j \min_{x \in F_j} x_i.$$

These bounds are independent of π if computed via $F = \bigcap_{j=1}^m F_j$.

(Order-Dependent Refinement): *If constraints are intersected sequentially (i.e., $F^{(k)}$ depends on π), then there exist permutations $\pi_1 \neq \pi_2$ such that:*

$$F_{\pi_1}^{(m)} \neq F_{\pi_2}^{(m)}.$$

Proof. Order-Independent Bounds. Same as Corollary B.2. The inequalities follow from $F \subseteq F_j$ for all j . Simultaneous intersection satisfies:

$$\max_{x \in F} x_i \leq \min_j \max_{x \in F_j} x_i \quad \text{and} \quad \min_{x \in F} x_i \geq \max_j \min_{x \in F_j} x_i,$$

as the intersection F cannot exceed the tightest bound from any F_j .

Order-Dependent Refinement. Let $F^{(k)} = \bigcap_{t=1}^k F_{\pi(t)}$. For dependent constraints (e.g., F_1 bounds x_1 , F_2 depends on x_1), the sequence $F^{(k)}$ depends on π . A counterexample with $F_1 : x_1 + x_2 \leq 2$ and $F_2 : x_1 - x_2 \leq 0$ shows:

1. Intersecting F_1 first gives $x_1 \leq 2 - x_2$, then F_2 further tightens $x_1 \leq x_2$.
2. Intersecting F_2 first gives $x_1 \leq x_2$, then F_1 tightens $x_1 \leq 1$.

The final bounds differ: $x_1 \leq \min(2 - x_2, x_2)$ vs. $x_1 \leq 1$. $\square$

C Algorithms

C.1 Overview

Fig. 3 shows the pipeline of our algorithm.

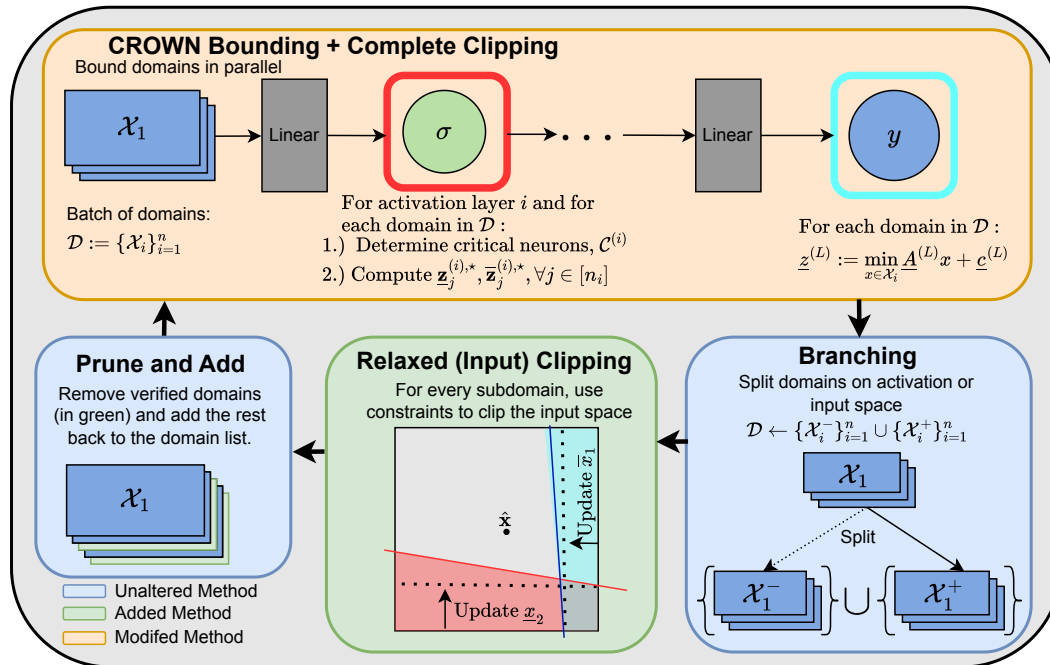


Figure 3: Full Clip-and-Verify pipeline for (Input and Activation Activation) BaB integration.

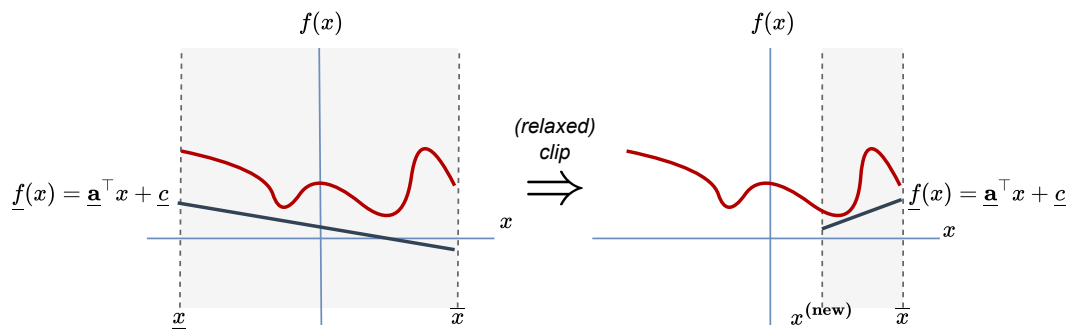


Figure 4: Simple 1D visualization of relaxed clipping reducing the input interval, potentially enabling a second pass of bound propagation to produce a tighter bound.

C.2 Sequential Clipping for Multiple Constraints

When several linear constraints are present, we apply Theorem 3.2 *sequentially* to each row of $\underline{A}x + \underline{c} \leq 0$. Algorithm 3 outlines this procedure: First, for each constraint k , compute $\hat{x}$ and ϵ from the *current* bounds, $\underline{x}$ and $\bar{x}$. Then, use Theorem 3.2 to refine $\underline{x}_i$ and $\bar{x}_i$ for all i . Finally, proceed to the next constraint, using the newly clipped bounds as the domain. Note that if $\underline{x}_i > \bar{x}_i$ occurs in any dimension, the input region of this subproblem is infeasible, and we can directly verify this subproblem without performing further verification.

Because the bounds are updated after each constraint, the final domain is an over-approximation of the true feasible region under all constraints simultaneously. However, it is still far more efficient than solving a multi-constraint system in one shot, making it well suited for large-scale verification where we repeatedly clip domains across many subproblems.

By clipping the domain repeatedly, Algorithm 3 retains enough precision to prune large regions yet remains computationally lightweight enough for repeated invocation on many subdomains during verification. It may even be the case that the constraints passed to our domain clipping algorithm reveal the region is entirely *infeasible*. Such a scenario may occur when two Activation assignments

Algorithm 3 Linear Constraint-Driven Relaxed Clipping (*sequential*)

Require: $\underline{x}$: Lower bounds; $\bar{x}$: Upper bounds; $\mathbf{A}$: Constraint matrix; $\mathbf{c}$: Constraint vector.

```
1:  $m \leftarrow \text{rows}(\mathbf{A}), n \leftarrow \text{cols}(\mathbf{A})$ 
2: for for each constraint  $k \in \{1, \dots, m\}$  do
3:   for each input dimension  $i \in \{1, \dots, n\}$  do
4:      $\hat{x} \leftarrow \frac{\bar{x} + \underline{x}}{2}, \epsilon \leftarrow \frac{\bar{x} - \underline{x}}{2}$ 
5:     if  $\mathbf{A}_{k,i} \neq 0$  then
6:        $\mathbf{x}_i^{(\text{new})} \leftarrow \frac{-\sum_{j \neq i} \mathbf{A}_{k,j} \hat{x}_j + \sum_{j \neq i} |\mathbf{A}_{k,j}| \epsilon_j - b}{\mathbf{A}_{k,i}}$ 
7:       if  $\mathbf{A}_{k,i} > 0$  then
8:          $\bar{x}_i^{(\text{new})} \leftarrow \min(\bar{x}_i, \mathbf{x}_i^{(\text{new})})$ 
9:       else
10:         $\underline{x}_i^{(\text{new})} \leftarrow \max(\underline{x}_i, \mathbf{x}_i^{(\text{new})})$ 
11:       $\underline{x}_i \leftarrow \max(\underline{x}_i, \mathbf{x}_i^{(\text{new})})$ 
12:       $\bar{x}_i \leftarrow \min(\bar{x}_i, \mathbf{x}_i^{(\text{new})})$ 
Ensure: Clipped  $\underline{x}, \bar{x}$ 
```

admit an infeasible domain, or when the desired property admits multiple constraints that produce infeasibility. In such scenarios, Algorithm 3 will return clipped bounds where $\underline{x}$ will be smaller than $\bar{x}$ along some dimension(s). In the context of branch-and-bound, this subdomain may be effectively pruned, avoiding the process of running bound propagation once more on the domain, avoiding unnecessary computations

C.3 Algorithms for Clip-and-Verify in Input Branch-and-bound Scheme

Algorithm 4 Clip-and-Verify for Input Branch-and-bound

Require: f : model to verify; n : batch size; timeout : time-out threshold

```
1:  $\mathcal{D}_{\text{Unknown}}, \underline{f} \leftarrow \text{Init}(f, \emptyset)$  {Initialize the set of unknown subdomains  $\mathcal{D}_{\text{Unknown}}$  and global bound  $\underline{f}$ }
2: while  $|\mathcal{D}_{\text{Unknown}}| > 0$  and not timed out do
3:    $\{\mathcal{X}_i, \underline{\mathbf{A}}_i^{\text{prev}}, \underline{\mathbf{c}}_i^{\text{prev}}\}_{i=1}^n \leftarrow \text{Batch\_Pick\_Out}(\mathcal{D}_{\text{Unknown}}, n)$  {Pick up to  $n$  subdomains  $\mathcal{X}_i$  and their associated hyperplanes  $(\underline{\mathbf{A}}_i^{\text{prev}}, \underline{\mathbf{c}}_i^{\text{prev}})$  from the previous iteration.}
4:    $\{\mathcal{C}_i\}_{i=1}^n \leftarrow \text{Top\_K\_Heuristic}(\{\mathcal{X}_i, \underline{\mathbf{A}}_i^{\text{prev}}, \underline{\mathbf{c}}_i^{\text{prev}}\}_{i=1}^n)$  {Determine critical neurons  $\mathcal{C}_i$  using a top-k heuristic over each domain and its previous hyperplanes.}
5:    $(\underline{f}_{\mathcal{X}_1}, \underline{\mathbf{A}}_{\mathcal{X}_1}, \underline{\mathbf{c}}_{\mathcal{X}_1}, \dots, \underline{f}_{\mathcal{X}_n}, \underline{\mathbf{A}}_{\mathcal{X}_n}, \underline{\mathbf{c}}_{\mathcal{X}_n}) \leftarrow \text{Solve\_Bound}(f, \{\mathcal{X}_i, \mathcal{C}_i\}_{i=1}^n)$  {Compute bounds and plane coefficients on each subdomain; Refine critical neurons using planes as constraints and Complete Clipping.}
6:    $\{\mathcal{X}_i^-, \mathcal{X}_i^+, \underline{\mathbf{A}}_i, \underline{\mathbf{c}}_i\}_{i=1}^n \leftarrow \text{Batch\_Split}(\{\mathcal{X}_i, \underline{\mathbf{A}}_i, \underline{\mathbf{c}}_i\}_{i=1}^n)$  {Split each subdomain; share hyperplanes with both children.}
7:    $\{\mathbf{x}_i^{(-)}, \mathbf{x}_i^{(+)}\}_{i=1}^n \leftarrow \text{Relaxed\_Clipper}(\{\mathcal{X}_i^-, \mathcal{X}_i^+, \underline{\mathbf{A}}_i, \underline{\mathbf{c}}_i\}_{i=1}^n)$  {Refine input box on each child subdomain using the plane coefficients.}
8:    $\{\mathcal{X}_i^-, \mathcal{X}_i^+\}_{i=1}^n \leftarrow \text{Domain\_Update}(\{\mathbf{x}_i^{(-)}, \mathbf{x}_i^{(+)}\}_{i=1}^n)$  {Update each child subdomain's input bounds.}
9:    $\mathcal{D}_{\text{Unknown}} \leftarrow \mathcal{D}_{\text{Unknown}} \cup \text{Domain\_Filter}\left(\left[\underline{f}_{\mathcal{X}_1^-}, \mathcal{X}_1^-, \underline{\mathbf{A}}_1^{(-)}, \underline{\mathbf{c}}_1^{(-)}\right], \dots\right)$  {Filter out verified/infeasible subdomains. Retain unknowns and their bounding hyper-planes.}
```

Ensure: UNSAT if $|\mathcal{D}_{\text{Unknown}}| = 0$ else Unknown

Algorithm 4 describes our modifications to the standard BaB procedure. When all subproblems can be verified, then the verification problem is referred to as UNSAT (i.e., the complementary property, $\exists \mathbf{x} \in \mathcal{X}, f(\mathbf{x}) < 0$, is *unsatisfiable*), and the network is *safe* from counter-examples. Otherwise, it is insufficient to determine if the property is UNSAT without further refinement or falsification.

C.4 Algorithms for Clip-and-Verify in Activation Branch-and-bound Scheme

Algorithm 5 Clip-and-Verify for Activation Branch-and-bound

Require: f : model to verify; n : batch size; timeout : time-out threshold

- 1: $\mathcal{D}_{\text{Unknown}}, \underline{f} \leftarrow \text{Init}(f, \emptyset)$ {Initialize the set of unknown subdomains $\mathcal{D}_{\text{Unknown}}$ and global bound $\underline{f}$ }
- 2: $\{\underline{\mathbf{A}}^{(j)}, \underline{\mathbf{c}}^{(j)}, \overline{\mathbf{A}}^{(j)}, \overline{\mathbf{c}}^{(j)}, \text{unstable_neuron}(\text{idx})^{(j)}\}_{j=1}^J \leftarrow \text{Get_Constraints}(f)$ {Retrieve the set of coefficients and biases for each unstable neurons' upper and lower bound and get the indices of unstable neurons during Bound Propagation.}
- 3: $\text{Domain_Clipper} \leftarrow \text{Init_Clipper}(\{\underline{\mathbf{A}}^{(j)}, \underline{\mathbf{c}}^{(j)}, \overline{\mathbf{A}}^{(j)}, \overline{\mathbf{c}}^{(j)}, \text{idx}^{(j)}\}_{j=1}^J)$ {Initialize the Domain Clipper with the set of constraints information.}
- 4: **while** $|\mathcal{D}_{\text{Unknown}}| > 0$ **and** not timed out **do**
- 5: $\{\mathcal{Z}_i\}_{i=1}^n \leftarrow \text{Batch_Pick_Out}(\mathcal{D}_{\text{Unknown}}, n)$ {Pick at most n subdomains from the unknown set.}
- 6: $\{\mathcal{Z}_i^-, \mathcal{Z}_i^+\}_{i=1}^n \leftarrow \text{Batch_Split}(\{\mathcal{Z}_i\}_{i=1}^n)$ {Split each subdomain (e.g., Activation split or input split) into two child subdomains.}
- 7: $\{\mathcal{C}_i^{(-)}, \mathcal{C}_i^{(+)}\}_{i=1}^n \leftarrow \text{Top-K_Heuristic}(\{\mathcal{X}_i\}_{i=1}^n)$ {Determine critical neurons using a top-k heuristic (e.g. BaBSR) over each domain.}
- 8: $\{\mathbf{x}_i^{(-)}, \text{interm_bds}_i^{(-)}, \mathbf{x}_i^{(+)}, \text{interm_bds}_i^{(+)}\}_{i=1}^n \leftarrow \text{Domain_Clipper}(\{\mathcal{Z}_i^-, \mathcal{Z}_i^+, \mathcal{C}_i^{(-)}, \mathcal{C}_i^{(+)}\}_{i=1}^n)$ {Apply Relaxed Clipping to child subdomain's input and Complete Clipping on the critical neurons. Constraints are the Activation splits.}
- 9: $\{\mathcal{Z}_i^-, \mathcal{Z}_i^+\}_{i=1}^n \leftarrow \text{Domain_Update}(\{\mathbf{x}_i^{(-)}, \text{interm_bds}_i^{(-)}, \mathbf{x}_i^{(+)}, \text{interm_bds}_i^{(+)}\}_{i=1}^n)$ {Update each child subdomain's input and intermediate bounds.}
- 10: $(\underline{f}_{\mathcal{Z}_1^-}, \dots, \underline{f}_{\mathcal{Z}_n^+}) \leftarrow \text{Solve_Bound}(f, \{\mathbf{x}_i, \text{interm_bounds}_i, \mathcal{Z}_i\})$ {Compute bounds on each newly clipped subdomain using a bound propagation solver.}
- 11: $\mathcal{D}_{\text{Unknown}} \leftarrow \mathcal{D}_{\text{Unknown}} \cup \text{Domain_Filter}(\underline{f}_{\mathcal{Z}_1^-}, \dots)$ {Filter out verified/infeasible subdomains. Keep remaining unknown subdomains in $\mathcal{D}_{\text{Unknown}}$.}

Ensure: UNSAT if $|\mathcal{D}_{\text{Unknown}}| = 0$ else Unknown

Algorithm 5 begins by initializing the verification procedure. First, we call $\text{Init}(f, \emptyset)$ to obtain an empty set of partial Activation assignments (or subdomains) along with an initial global lower bound $\underline{f}$ for the property to be checked. This global lower bound can, for example, be the result of a quick bounding pass. The set $\mathcal{D}_{\text{Unknown}}$ is then populated with a single “root” subdomain representing the entire input domain.

Next, we retrieve the constraint information for all neurons (Line 2). Specifically, $\text{Get_Constraints}(f)$ returns the linear coefficients and biases used to bound each neuron's activation, distinguishing between the lower-bounding ($\underline{\mathbf{A}}^{(j)}, \underline{\mathbf{c}}^{(j)}$) and upper-bounding ($\overline{\mathbf{A}}^{(j)}, \overline{\mathbf{c}}^{(j)}$) linear functions. It also identifies indices of “unstable” Activation neurons whose ranges straddle zero. These constraints will later be used to restrict the feasible input region using both our Relaxed Clipping and Complete Clipping algorithms.

We then initialize the Domain Clipper (Line 3) with the gathered linear constraints. This Clipper component will be invoked whenever we branch on a Activation neuron, so that the corresponding partial assignment (e.g., $x_k^{(j)} \geq 0$) is “pushed back” onto the input domain. In doing so, we clip the subdomain's input box by applying Theorem 3.2 (or its extensions) to incorporate these newly introduced constraints, thus discarding parts of the input space that violate them. In addition, when given a set of critical neurons, these Activation split constraints will be used to directly refine the intermediate neurons using Theorem 3.1.

The main loop (Lines 5–11) iterates until either no subdomains remain unknown or a time-out is reached. In each iteration, we pick up to n unknown subdomains from $\mathcal{D}_{\text{Unknown}}$ (Line 5) for batched parallel processing. Each subdomain $\mathcal{Z}_i$ is then split (Line 6) along one or more unstable Activation neurons, creating child subdomains in which each split neuron is fixed to either the active (≥ 0) or inactive (≤ 0) regime.

At Line 7, we invoke a top-k heuristic (e.g. BaBSR) in order to determine the set of “critical neurons” that would contribute the most to providing a stronger convex relaxation if their bounds were to be refined.

At Lines 8–9, we invoke the Domain Clipper on these newly formed child subdomains. The Clipper translates each Activation assignment into a linear constraint on the input, then refines (or “clips”) the child subdomain’s input bounds, and the bounds of the “critical neurons” in each subdomain with respect to the heuristic choices from the step prior. This ensures that any portion of the parent domain that contradicts the new constraint is removed. Once clipped, the child subdomains’ intermediate bounds are also updated (Line 10) so that subsequent bounding calculations reflect the tighter input ranges.

We then compute bounds on the newly clipped subdomains (11) using a chosen method—often a fast bound-propagation tool such as CROWN or a lightweight LP solver. This step yields lower bounds $\underline{f}_{\mathcal{Z}_i^\pm}$ on the network outputs for each subdomain. If these bounds confirm that the property holds (e.g., a robustness margin remains non-negative), the subdomain is verified and can be pruned. If the subdomain is infeasible (e.g., constraints are contradictory), it is also removed. All remaining subdomains (still “unknown”) return to $\mathcal{D}_{\text{Unknown}}$ for further splitting.

Finally, the loop terminates once there are no unknown subdomains left or the time-out is reached. If $\mathcal{D}_{\text{Unknown}}$ becomes empty, we conclude UNSAT, signifying that no violating input (counterexample) exists within any subdomain. Otherwise, we return “Unknown,” indicating that verification was not completed in time.

Overall, this procedure reflects a standard Activation BaB flow, except that an additional “domain clipping” step (Lines 8–9) is inserted after each split, leveraging partial Activation assignments to refine the input domain and select intermediate neurons before the next bounding pass. By applying our linear constraint-driven clipping algorithms whenever new constraints appear, we gain significantly tighter intermediate-layer bounds and thus reduce the branching burden throughout the verification process.

C.5 2D Toy Example

To illustrate our clipping verification approach, we consider a simple two-layer ReLU feed-forward network defined as

$$f(\mathbf{x}) = \mathbf{w}^{(2)\top} \sigma(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)}), \quad (22)$$

where $\mathbf{w}^{(2)}, \mathbf{b}^{(1)}, \mathbf{x} \in \mathbb{R}^2$, $\mathbf{W}^{(1)} \in \mathbb{R}^{2 \times 2}$, and $\sigma(\cdot)$ denotes the element-wise ReLU activation. We aim to verify the property

$$f(\mathbf{x}) \geq 0, \quad \forall \mathbf{x} \in \mathcal{X}.$$

The input domain $\mathcal{X}$ is defined as an ℓ_∞ -box centered at $\hat{\mathbf{x}} = [0.5, -0.5]^\top$ with half-widths $\boldsymbol{\epsilon} = [1.5, 1.5]^\top$. Equivalently, the domain can be expressed using its endpoints $\underline{\mathbf{x}} = [-1, -2]^\top$ and $\bar{\mathbf{x}} = [2, 1]^\top$. The network parameters are specified as

$$\left\{ \mathbf{W}^{(1)} = \begin{bmatrix} 1 & -7 \\ 5 & -1 \end{bmatrix}, \quad \mathbf{b}^{(1)} = \begin{bmatrix} 6 \\ -7 \end{bmatrix}, \quad \mathbf{w}^{(2)} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right\}. \quad (23)$$

CROWN Bound. As a warm up, let’s bound the network using the CROWN algorithm without utilizing relaxed nor complete clipping. We begin by computing the pre-activation bounds of the intermediate layer which are obtained by concretizing the first layer’s affine transformation:

$$\underline{\mathbf{z}} = \min_{\mathbf{x} \in \mathcal{X}} \mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)}, \quad \bar{\mathbf{z}} = \max_{\mathbf{x} \in \mathcal{X}} \mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)}. \quad (24a)$$

$$\underline{\mathbf{z}} = \mathbf{W}^{(1)} \hat{\mathbf{x}} - |\mathbf{W}^{(1)}| \boldsymbol{\epsilon} + \mathbf{b}^{(1)}, \quad \bar{\mathbf{z}} = \mathbf{W}^{(1)} \hat{\mathbf{x}} + |\mathbf{W}^{(1)}| \boldsymbol{\epsilon} + \mathbf{b}^{(1)}. \quad (24b)$$

Substituting the parameters into equation (24b) yields $\underline{\mathbf{z}} = [-2, -13]^\top$ and $\bar{\mathbf{z}} = [22, 5]^\top$. These neurons are then passed to a ReLU activation function, and one should notice that for each input, we have that $\underline{z}_1 < 0 < \bar{z}_1$ and $\underline{z}_2 < 0 < \bar{z}_2$. In this scenario, we have that the ReLU neurons are *unstable*, meaning that we cannot bound the non-linearity exactly, however, we can still get sound linear bounds on the output of the ReLU activation. Using Lemma A.1, we construct the diagonal matrices and corresponding vectors,

$$\underline{\mathbf{D}} = \begin{bmatrix} \alpha_1 & 0 \\ 0 & \alpha_2 \end{bmatrix}, \quad \underline{\mathbf{b}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (25a)$$

$$\overline{\mathbf{D}} = \begin{bmatrix} \frac{\bar{z}_1}{\bar{z}_1 - \underline{z}_1} & 0 \\ 0 & \frac{\bar{z}_2}{\bar{z}_2 - \underline{z}_2} \end{bmatrix}, \quad \bar{\mathbf{b}} = \begin{bmatrix} \frac{-\bar{z}_1 \underline{z}_1}{\bar{z}_1 - \underline{z}_1} \\ \frac{-\bar{z}_2 \underline{z}_2}{\bar{z}_2 - \underline{z}_2} \end{bmatrix}. \quad (25b)$$

$\underline{\mathbf{D}}$ and $\underline{\mathbf{b}}$ are used to create *lower bounding* planes on the output of the ReLU activation where α_1 and α_2 are real numbers limited to the range $[0, 1]$. These values may be optimized, however we will always fix $\alpha_1 = \alpha_2 = 1$ when lower bounding unstable neurons in this toy example. $\overline{\mathbf{D}}$ and $\bar{\mathbf{b}}$ are *upper bounding* planes on the output of the ReLU activation, and its construction is derived from the Planet relaxation [25]. It may be verified that for inputs $\mathbf{z}$ in the range $[\underline{\mathbf{z}}, \bar{\mathbf{z}}]$,

$$\overline{\mathbf{D}}\mathbf{z} + \bar{\mathbf{b}} \geq \sigma(\mathbf{z}) \geq \underline{\mathbf{D}}\mathbf{z} + \underline{\mathbf{b}}. \quad (26)$$

We next lower bound the network output. Because the post-activation vector $\sigma(\mathbf{z})$ is passed through the final linear layer, a coordinate-wise sign on the final-layer weights determines whether to use upper or lower affine bounds for each neuron. Concretely,

$$\begin{cases} w\sigma(\mathbf{z})_i \geq w(\underline{\mathbf{D}}_{i,i}\mathbf{z}_i + \underline{\mathbf{b}}_i), & \text{if } w \geq 0 \\ w\sigma(\mathbf{z})_i \geq w(\overline{\mathbf{D}}_{i,i}\mathbf{z}_i + \bar{\mathbf{b}}_i), & \text{if } w < 0 \end{cases} \quad (27)$$

so that the final-layer lower bound with respect to $\mathbf{z}$ is

$$\mathbf{w}^{(2)\top} \sigma(\mathbf{z}) \geq (\mathbf{w}^{(2),+})^\top (\underline{\mathbf{D}}\mathbf{z} + \underline{\mathbf{b}}) + (\mathbf{w}^{(2),-})^\top (\overline{\mathbf{D}}\mathbf{z} + \bar{\mathbf{b}}) \quad (28)$$

where $\mathbf{w}^{(2),+}$ zeros out entries which are negative and $\mathbf{w}^{(2),-}$ zeros out entries which are positive. In our example, these vectors are $\mathbf{w}^{(2),+} = [1, 0]^\top$ and $\mathbf{w}^{(2),-} = [0, -1]^\top$. The final step is to produce a lower bounding hyperplane of the final layer with respect to the network's input. So far, we've only related this lower bound to the intermediate input, $\mathbf{z}$. Our final step is quite simple as we know that $\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$. Using this relation, we can “back-propagate” our affine relaxations to the input as follows,

$$(\mathbf{w}^{(2),+})^\top (\underline{\mathbf{D}}\mathbf{z} + \underline{\mathbf{b}}) + (\mathbf{w}^{(2),-})^\top (\overline{\mathbf{D}}\mathbf{z} + \bar{\mathbf{b}}) \quad (29a)$$

$$= (\mathbf{w}^{(2),+})^\top \left(\underline{\mathbf{D}} \left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \right) + \underline{\mathbf{b}} \right) + (\mathbf{w}^{(2),-})^\top \left(\overline{\mathbf{D}} \left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \right) + \bar{\mathbf{b}} \right) \quad (29b)$$

$$= \left(\left((\mathbf{w}^{(2),+})^\top \underline{\mathbf{D}} + (\mathbf{w}^{(2),-})^\top \overline{\mathbf{D}} \right) \mathbf{W}^{(1)} \right) \mathbf{x} + \left((\mathbf{w}^{(2),+})^\top \underline{\mathbf{D}} + (\mathbf{w}^{(2),-})^\top \overline{\mathbf{D}} \right) \mathbf{b}^{(1)} + (\mathbf{w}^{(2),+})^\top \underline{\mathbf{b}} + (\mathbf{w}^{(2),-})^\top \bar{\mathbf{b}} \quad (29c)$$

Let us introduce the following variables,

$$\begin{cases} \underline{\mathbf{a}} := \left((\mathbf{w}^{(2),+})^\top \underline{\mathbf{D}} + (\mathbf{w}^{(2),-})^\top \overline{\mathbf{D}} \right) \mathbf{W}^{(1)} \\ \underline{\mathbf{c}} := \left((\mathbf{w}^{(2),+})^\top \underline{\mathbf{D}} + (\mathbf{w}^{(2),-})^\top \overline{\mathbf{D}} \right) \mathbf{b}^{(1)} + (\mathbf{w}^{(2),+})^\top \underline{\mathbf{b}} + (\mathbf{w}^{(2),-})^\top \bar{\mathbf{b}} \end{cases} \quad (30)$$

By construction, we have that the output of the network is lower bounded by the following linear relaxation,

$$f(\mathbf{x}) \geq \underline{\mathbf{a}}^\top \mathbf{x} + \underline{\mathbf{c}}, \quad \mathbf{x} \in \mathcal{X}. \quad (31)$$

Using Hölder's inequality, we can solve for the minima of this lower bounding plane,

$$\min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) \geq \min_{\mathbf{x} \in \mathcal{X}} \underline{\mathbf{a}}^\top \mathbf{x} + \underline{\mathbf{c}} = \underline{\mathbf{a}}^\top \hat{\mathbf{x}} - |\underline{\mathbf{a}}|^\top \epsilon + \underline{\mathbf{c}} = -\frac{19}{6}. \quad (32)$$

As this lower bound is too loose, we cannot verify our desired property, $f(\mathbf{x}) \geq 0, \forall \mathbf{x} \in \mathcal{X}$.

Relaxed Clipping. We first tackle this example using our efficient yet approximate clipping algorithm termed *relaxed clipping*. Given a set of constraints, the objective of relaxed clipping is to compute the smallest input box representation that satisfies those constraints. This step is performed once per round of branch-and-bound, and the resulting refined input box is shared by all neurons during the concretization step. Consequently, this refinement has the potential to tighten bounds across multiple neurons, including those in the final layer.

Consider a bound-propagation-based verifier executing branch-and-bound by splitting over the activation space. For the remainder of this subsection, we focus on the case where the first ReLU

neuron is split into its non-positive (inactive) region, i.e., $z_1 \leq 0$. This split decision can be reinterpreted as a linear constraint on the input,

$$(\mathbf{W}_{1,:}^{(1)})^\top \mathbf{x} + \mathbf{b}_1^{(1)} \leq 0. \quad (33)$$

Since the input $\mathbf{x}$ is two-dimensional, our goal is to tighten the lower and upper bounds of each input dimension under this constraint. Formally, we solve the following optimization problems:

$$\underline{x}_1^{(rc)} := \min_{\mathbf{x} \in \mathcal{X}} x_1 \quad \text{s.t.} \quad (\mathbf{W}_{1,:}^{(1)})^\top \mathbf{x} + \mathbf{b}_1^{(1)} \leq 0, \quad (34a)$$

$$\bar{x}_1^{(rc)} := \max_{\mathbf{x} \in \mathcal{X}} x_1 \quad \text{s.t.} \quad (\mathbf{W}_{1,:}^{(1)})^\top \mathbf{x} + \mathbf{b}_1^{(1)} \leq 0, \quad (34b)$$

$$\underline{x}_2^{(rc)} := \min_{\mathbf{x} \in \mathcal{X}} x_2 \quad \text{s.t.} \quad (\mathbf{W}_{1,:}^{(1)})^\top \mathbf{x} + \mathbf{b}_1^{(1)} \leq 0, \quad (34c)$$

$$\bar{x}_2^{(rc)} := \max_{\mathbf{x} \in \mathcal{X}} x_2 \quad \text{s.t.} \quad (\mathbf{W}_{1,:}^{(1)})^\top \mathbf{x} + \mathbf{b}_1^{(1)} \leq 0. \quad (34d)$$

From Theorem 3.2, one potential solution along each dimension is given by

$$\mathbf{x}_i^{(clip)} := \frac{-\sum_{j \neq i} \left\{ \mathbf{W}_{1,j}^{(1)} \hat{\mathbf{x}}_j - |\mathbf{W}_{1,j}^{(1)}| \epsilon_j \right\} - \mathbf{b}_1^{(1)}}{\mathbf{W}_{1,i}^{(1)}} \quad (35)$$

Each dimension is then updated as follows:

$$\begin{cases} \bar{x}_i^{(rc)} = \min \left\{ \mathbf{x}_i^{(clip)}, \bar{x}_i \right\} & \text{if } \mathbf{W}_{1,i}^{(1)} > 0 \\ \underline{x}_i^{(rc)} = \max \left\{ \mathbf{x}_i^{(clip)}, \underline{x}_i \right\} & \text{if } \mathbf{W}_{1,i}^{(1)} < 0 \\ \text{no change} & \text{otherwise} \end{cases} \quad (36)$$

For the first dimension ($i = 1$), since $\mathbf{W}_{1,1}^{(1)} > 0$, the upper limit may be refined if $\mathbf{x}_1^{(clip)} < \bar{x}_1$. For the second dimension ($i = 2$), where $\mathbf{W}_{1,2}^{(1)} < 0$, the lower limit may be refined if $\mathbf{x}_2^{(clip)} > \underline{x}_2$. Substituting the given parameters yields:

$$\mathbf{x}_1^{(clip)} = \frac{-\mathbf{W}_{1,2}^{(1)} \hat{\mathbf{x}}_2 + |\mathbf{W}_{1,2}^{(1)}| \epsilon_2 - \mathbf{b}_1^{(1)}}{\mathbf{W}_{1,1}^{(1)}} = \frac{-(-7)(-1/2) + |-7|(3/2) - 6}{1} = 1 \quad (37a)$$

$$\mathbf{x}_2^{(clip)} = \frac{-\mathbf{W}_{1,1}^{(1)} \hat{\mathbf{x}}_1 + |\mathbf{W}_{1,1}^{(1)}| \epsilon_1 - \mathbf{b}_1^{(1)}}{\mathbf{W}_{1,2}^{(1)}} = \frac{-(1)(1/2) + |1|(3/2) - 6}{-7} = \frac{5}{7} \quad (37b)$$

It is indeed the case that $\mathbf{x}_1^{(clip)} < \bar{x}_1$ and $\mathbf{x}_2^{(clip)} > \underline{x}_2$, so the limits of the refined input domain are now,

$$\underline{\mathbf{x}}^{(rc)} = [-1, 5/7]^\top, \quad \bar{\mathbf{x}}^{(rc)} = [1, 1]^\top. \quad (38)$$

The new box representation may also be characterized by its center and half-widths:

$$\hat{\mathbf{x}}^{(rc)} = (\bar{\mathbf{x}}^{(rc)} + \underline{\mathbf{x}}^{(rc)})/2, \quad \boldsymbol{\epsilon}^{(rc)} = (\bar{\mathbf{x}}^{(rc)} - \underline{\mathbf{x}}^{(rc)})/2, \quad (39)$$

yielding the refined input domain

$$\mathcal{X}^{(rc)} = \left\{ \mathbf{x} \mid \|\hat{\mathbf{x}}^{(rc)} - \mathbf{x}\|_\infty \leq \boldsymbol{\epsilon}^{(rc)} \right\} = \left\{ \mathbf{x} \mid \underline{\mathbf{x}}^{(rc)} \leq \mathbf{x} \leq \bar{\mathbf{x}}^{(rc)} \right\}. \quad (40)$$

We now proceed to bound the network using the CROWN algorithm under $\mathcal{X}^{(rc)}$ in place of the original domain, $\mathcal{X}$. Substituting this domain into Eq. (24) produces $\underline{\mathbf{z}}^{(rc)} = [-2, -13]^\top$ and $\bar{\mathbf{z}}^{(rc)} = [2, -57/21]^\top$. Both neurons exhibit tighter bounds, and notably, $\bar{z}_2^{(rc)} < 0$, which ensures that the input to the second ReLU neuron is strictly non-positive across $\mathcal{X}^{(rc)}$. Since the ReLU function is piecewise linear, this implies an *exact* post-activation bound:

$$\underline{\sigma}(\mathbf{z}) = \sigma(\mathbf{z}) = \bar{\sigma}(\mathbf{z}) = 0, \quad \forall \mathbf{x} \in \mathcal{X}^{(rc)}. \quad (41)$$

One interesting observation to point out is that with this clipped domain, $\bar{z}_1^{(rc)} = 2$ improves dramatically from the original upper bound, $\bar{z}_1 = 22$. However, because our constraint originated

from $z_1 \leq 0$, the true maximum is in fact zero. In this case, one could enforce this upper bound to be zero, but keep in mind that relaxed clipping algorithm is designed for generality, thus constraints may arise from split decisions at any neuron in the network where forcing $\bar{z}_1^{(rc)} = 0$ would not be valid. This example illustrates that relaxed clipping, while lightweight and compatible with CROWN, remains an approximation and may yield suboptimal yet informative bounds. Rather than enforcing the true maximum, we retain the relaxed clipping result, $\bar{z}_1^{(rc)} = 2$.

Applying Lemma A.1, we derive the linear post-activation bounds with respect to the pre-activation inputs:

$$\underline{\mathbf{D}}^{(rc)} = \begin{bmatrix} \alpha_1 & 0 \\ 0 & 0 \end{bmatrix}, \quad \underline{\mathbf{b}}^{(rc)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (42a)$$

$$\overline{\mathbf{D}}^{(rc)} = \begin{bmatrix} \frac{\bar{z}_1^{(rc)}}{\bar{z}_1^{(rc)} - z_1^{(rc)}} & 0 \\ 0 & 0 \end{bmatrix}, \quad \overline{\mathbf{b}}^{(rc)} = \begin{bmatrix} \frac{-\bar{z}_1^{(rc)} z_1^{(rc)}}{\bar{z}_1^{(rc)} - z_1^{(rc)}} \\ 0 \end{bmatrix}. \quad (42b)$$

For simplicity, let $\alpha_1 = 1$. Notice that for the second neuron, we have that,

$$(\underline{\mathbf{D}}_{2,:}^{(rc)})^\top \mathbf{z} + \underline{\mathbf{b}}_2^{(rc)} = \sigma(z)_2 = (\overline{\mathbf{D}}_{2,:}^{(rc)})^\top \mathbf{z} + \overline{\mathbf{b}}_2^{(rc)} = 0, \quad \forall z_2 \in [z_2^{(rc)}, \bar{z}_2^{(rc)}], \quad (43)$$

confirming that this ReLU neuron is inactive and its output can be exactly bounded as zero.

Finally, performing the same backpropagation procedure as before yields

$$\min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) \geq \min_{\mathbf{x} \in \mathcal{X}^{(rc)}} \underline{\mathbf{a}}^{(rc)\top} \mathbf{x} + \underline{\mathbf{c}}^{(rc)} = \underline{\mathbf{a}}^{(rc)\top} \hat{\mathbf{x}}^{(rc)} - |\underline{\mathbf{a}}^{(rc)}|^\top \boldsymbol{\epsilon}^{(rc)} + \underline{\mathbf{c}}^{(rc)} = -2. \quad (44)$$

This represents an improvement over the previous bound, although the property remains unverified.

Complete Clipping. Relaxed clipping can be viewed as an *indirect* approach towards refining the bounds of the neural network as a smaller input representation can potentially yield improvement in the concretization step when forming the neurons' bounds. *Complete clipping* on the other hand is a *direct* approach towards refining the bounds on the network as the optimization objective specifically targets the neuron's bounds rather than the shared input representation.

In practice, relaxed clipping is extremely lightweight, and its operations are well-suited for GPUs. Thus, it is often sensible to combine relaxed and complete clipping in the verification pipeline. In this example, however, we aim to clearly distinguish the two methods. Therefore, we use the original input domain, $\mathcal{X}$, rather than the refined one, $\mathcal{X}^{(rc)}$, and retain the ReLU split constraint $z_1 \leq 0$.

Complete clipping operates directly on the bounds of each neuron, refining them via constrained optimization. The first step is to bound the preactivation bounds which may be formulated as:

$$\begin{aligned} \underline{z}_i^{(cc)} &= \min_{\mathbf{x} \in \mathcal{X}} \mathbf{W}_{i,:}^{(1)} \mathbf{x} + \mathbf{b}_i^{(1)} & \bar{z}_i^{(cc)} &= \max_{\mathbf{x} \in \mathcal{X}} \mathbf{W}_{i,:}^{(1)} \mathbf{x} + \mathbf{b}_i^{(1)} \\ \text{s.t. } \left(\mathbf{W}_{1,:}^{(1)} \right)^\top \mathbf{x} + \mathbf{b}_1^{(1)} &\leq 0 & \text{s.t. } \left(\mathbf{W}_{1,:}^{(1)} \right)^\top \mathbf{x} + \mathbf{b}_1^{(1)} &\leq 0 \end{aligned} \quad (45a) \quad (45b)$$

For this two-dimensional problem, this amounts to targeting the lower and upper bound of both neurons at the intermediate layer, resulting in a total of four constrained optimization subproblems. However, we will only focus on refining the upper bounds as we will soon see that this will be sufficient for verifying our desired property in this toy example.

As discussed earlier, the split constraint on the first neuron implies that its true upper bound is trivially $\bar{z}_1^* = 0$. With a single constraint, Theorem 3.1 guarantees optimality, i.e., $\bar{z}_1^{(cc)} = \bar{z}_1^* = 0$, which can be verified using Algorithm 1². For a single constraint, complete clipping attains the exact optimal solution through enumeration, without relying on projected gradient methods or LP solvers, no longer serving as an approximation such as the case with relaxed clipping. Even when multiple constraints are present (where optimality is not guaranteed), this targeted refinement remains a powerful mechanism for improving intermediate-layer bounds.

²Algorithm 1 performs coordinate ascent on the dual objective when the primal is a *minimization* problem. Since (45b) is a *maximization* problem, we can negate the primal objective to minimize it, then negate the resulting solution. Furthermore, the algorithm assumes constraints of the form $\mathbf{G}\mathbf{x} + \mathbf{h} \leq \mathbf{0}$. For constraints of the opposite form, $\mathbf{G}\mathbf{x} + \mathbf{h} \geq \mathbf{0}$, one may simply negate them. Hence, the algorithm is used without loss of generality.

Next, we consider the upper bound $\bar{z}_2^{(cc)}$. Complete clipping yields the optimal solution $\bar{z}_2^*$, which we derive analytically via Algorithm 1. According to Theorem 3.1, the dual form of this problem and its solution is given by

$$L^* = \min_{\beta \in \mathbb{R}^+} \left(\left(\mathbf{W}_{2,:}^{(1)} - \beta \mathbf{W}_{1,:}^{(1)} \right)^\top \hat{\mathbf{x}} + \sum_{j=1}^n \left| \mathbf{W}_{2,j}^{(1)} - \beta \mathbf{W}_{1,j}^{(1)} \right| \epsilon_j + \mathbf{b}_2^{(1)} - \beta \mathbf{b}_1^{(1)} \right). \quad (46)$$

The dual objective is minimized with respect to the Lagrange multiplier, $\beta \in \mathbb{R}^+$. One possible solution occurs at $\beta = 0$, but since relaxed clipping already improved this bound, we expect that $\beta^* \neq 0$.

Before solving for β^* , it is helpful to examine the structure of the dual objective $D(\beta)$. Because the primal objective maximizes z_2 , minimizing the dual objective is equivalent. The function $D(\beta)$ is *convex* and *piece-wise linear*, so we analyze its sub-gradient, $\frac{\partial}{\partial \beta} D(\beta)$:

$$\begin{cases} \left(-\mathbf{W}_{1,:}^{(1)} \right)^\top \hat{\mathbf{x}} + \sum_{j=1}^2 \left\{ \text{sign} \left(\mathbf{W}_{2,j}^{(1)} - \beta \mathbf{W}_{1,j}^{(1)} \right) (-\mathbf{W}_{1,j}^{(1)}) \epsilon_j \right\} - \mathbf{b}_1^{(1)} & , \beta \notin \mathbf{q} \\ \left[\frac{\partial}{\partial \beta} D(\beta^-), \frac{\partial}{\partial \beta} D(\beta^+) \right] & , \beta \in \mathbf{q} \end{cases} \quad (47)$$

where $\mathbf{q}$ is the vector of breakpoints $[(\mathbf{W}_{2,j}^{(1)})/(\mathbf{W}_{1,j}^{(1)})]_{j=1}^2 = [5, 1/7]$. Thus, the gradient is uniquely defined on the intervals $\beta \in (-\infty, 1/7)$, $(1/7, 5)$, and $(5, \infty)$. Because $D(\beta)$ is convex, its sub-gradient is negative on the leftmost interval and positive on the rightmost one. The optimal β^* occurs at the break-point where the sub-gradient changes sign from negative to positive, i.e., the break-point whose sub-gradient interval contains zero. The sub-gradients in each region are:

$$\begin{cases} -|\mathbf{W}_{1,1}^{(1)}| \epsilon_1 - |\mathbf{W}_{1,2}^{(1)}| \epsilon_2 - \left(\mathbf{W}_{1,:}^{(1)} \right)^\top \hat{\mathbf{x}} - \mathbf{b}_1^{(1)} = -24 & , \beta < \frac{1}{7} \\ -|\mathbf{W}_{1,1}^{(1)}| \epsilon_1 + |\mathbf{W}_{1,2}^{(1)}| \epsilon_2 - \left(\mathbf{W}_{1,:}^{(1)} \right)^\top \hat{\mathbf{x}} - \mathbf{b}_1^{(1)} = -1 & , \frac{1}{7} < \beta < 5 \\ +|\mathbf{W}_{1,1}^{(1)}| \epsilon_1 + |\mathbf{W}_{1,2}^{(1)}| \epsilon_2 - \left(\mathbf{W}_{1,:}^{(1)} \right)^\top \hat{\mathbf{x}} - \mathbf{b}_1^{(1)} = 2 & , \beta > 5. \end{cases} \quad (48)$$

Note that when transitioning from the case where $\beta \in (-\infty, 1/7)$ to $\beta \in (1/7, 5)$, the term $|\mathbf{W}_{1,2}^{(1)}| \epsilon_2$ switches sign while $-|\mathbf{W}_{1,1}^{(1)}| \epsilon_1$ remains negative. This ordering follows because in our break-point vector, we have that $q_2 < q_1$, and this ensures the sub-gradient is correctly calculated in each sub-interval, providing intuition as to why $\text{argsort}(\mathbf{q})$ is necessary in Algorithm 1. For this example, the sign change occurs at $\beta = 5$, giving the minimum of the dual objective at $\beta^* = 5$ and the solution $\bar{z}^{(cc)} = \bar{z}_2^* = L^* = -3$.

After performing complete clipping at the intermediate layer, we discover that the optimal upper bounds under our split constraint are given as $\bar{\mathbf{z}}^* = [0, -3]^\top$. There is clearly an inter-neuron dependency between the two ReLU neurons such that forcing the first neuron to be in-active subsequently causes the the second neuron to also become in-active. Consequently, the post-activation neurons can be *exactly* bounded using the CROWN algorithm, and in particular, $\sigma(\mathbf{z}) = \mathbf{0}$ for all $\mathbf{z} \in [\underline{\mathbf{z}}^*, \bar{\mathbf{z}}^*]$. Since the last layer contains no bias vector, it is not necessary to perform complete clipping again, and we have finally verified our desired property for this subproblem,

$$\min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) = \mathbf{w}^{(2)\top} \sigma(\mathbf{z}) \geq 0, \quad \text{subject to } \mathbf{z}_1 \leq 0. \quad (49)$$

D Experiments

D.1 Experiments Settings

To allow for comparability of results, all tools for input BaB were evaluated on equal-cost hardware with a 32-vcore CPU, one NVIDIA RTX 4090 GPU with 24 GB memory, and 256 GB CPU memory. For ReLU based BaB experiment, we use a cluster with one AMD EPYC 9534 64-core CPU and the GPU is one NVIDIA RTX 5090 GPU with 32 GB memory and 512 GB CPU memory. Our implementation is based on the open-source α, β -CROWN verifier³ with Clip-and-Verify related

³<https://github.com/huanzhang12/alpha-beta-CROWN>

code added. For input bab, three different set-ups of Clip-and-Verify are tested: Relaxed, Relaxed + Reorder, and Complete. Here Relaxed, Reorder and Complete refers to the methodology discussed in 3.2 and 3.3. All experiments use 32 CPU cores and 1 GPU. The MIP cuts are acquired by the cplex [34] solver (version 22.1.0.0). We use the Adam optimizer [38] to solve both α, β, μ, τ . For the SDP-FO benchmarks, we optimize those parameters for 20 iterations with a learning rate of 0.1 for α and 0.02 for β, μ, τ . We decay the learning rates with a factor of 0.98 per iteration. The timeout is 200s per instance. For the VNN-COMP benchmarks, we use the same configuration as α, β -CROWN used in the respective competition and the same timeouts. For NN control systems, task details are in [42, Appendix C]. For the Lyapunov function level set, we verify on $V(x) \in [0.2, 0.20001]$ for CartPole and Quadrotor-2D, and on $V(x) \in [2.0, 2.1]$ for Quadrotor-2D-Large-ROA. Let Ω_{final} denote the resulting (expanded) box used for training/evaluation once the generator stabilizes. Concretely (all “ $\pm[\cdot]$ ” are per-coordinate half-widths), for CartPole, $\Omega_{\text{final}} = \pm[4.8, 3.6, 13.2, 13.2]$. For Quadrotor-2D, $\Omega_{\text{final}} = \pm[12, 13.2, 12, 19.2, 20.4, 88.8]$.

D.2 Ablation Studies

We conduct a detailed ablation study on multiple adversarially-trained models spanning MNIST, CIFAR, and larger VNN-COMP benchmarks to evaluate Clip-and-Verify and its variants against three baselines: β -CROWN, GCP-CROWN, and BICCOS. Tables 4 and 6 highlight three key metrics: verified accuracy (Ver.%), average per-example verification time (Time), and average per-verified-example domain visited (D.V.). Domain visited (D.V.) is a metric specific to branch-and-bound (BaB) methods, indicating how many subproblems (domains) are explored to fully verify an instance. Crucially, a higher D.V. count may reflect verifying more difficult instances or a larger overall coverage, rather than inefficiency in the verification process. We also provide Figures 5 and 6 that visualize these metrics across all benchmarks.

Overall Verified Accuracy and Time From Table 4, Clip-and-Verify variants nearly always achieve higher verified accuracy than the baselines. For instance, on CNN-B-Adv (CIFAR), Clip-and-Verify with BICCOS reaches 51.5% verified accuracy—surpassing the 47.0% (β -CROWN), 49.5% (GCP-CROWN with MIP cuts), and 51.0% (BICCOS alone) of the baselines. On cifar10-resnet, Clip-and-Verify (with MIP cuts or with BICCOS) achieves up to 88.89% verified accuracy (outperforming the 83.33% - 87.5% range from baselines). In terms of verification time on this benchmark, Clip-and-Verify with β -CROWN is the fastest overall (6.06s), and Clip-and-Verify with BICCOS (11.80s) remains competitive with standalone BICCOS (16.73s) and GCP-CROWN (17.99s). When scaling to deeper networks such as cifar100-2024 and tinyimagenet-2024, Clip-and-Verify with BICCOS maintains the leading verified accuracy (65.5% and 72.0%, respectively) while sustaining moderate average verification times (e.g., 8.17s for cifar100-2024 and 10.48s for tinyimagenet-2024), underscoring its suitability for larger-scale verification tasks.

Domain Visited (D.V.) vs. Difficulty In Table 6, we further examine the average domain visited (D.V.) across verified examples. While Clip-and-Verify variants may sometimes visit more domains (e.g., on oval22, Clip-and-Verify with MIP cuts visits 18891.25 domains compared to 16614.72 for GCP-CROWN with MIP cuts), this often correlates with achieving higher verified accuracy (90.00% vs 83.33% in this case). This suggests that the method is effectively exploring the space to verify more challenging instances or a broader set of inputs, leading to a net increase in verified accuracy. For example, on CNN-A-Adv (CIFAR), Clip-and-Verify with MIP cuts visits 3704.86 domains on average and attains 48.5% verified accuracy. While its D.V. is slightly higher than standalone BICCOS (3622.71 D.V. for 48.5% accuracy), it’s notably lower than β -CROWN (12621.38 D.V. for 45.5% accuracy) and GCP-CROWN with MIP cuts (8186.11 D.V. for 48.5% accuracy), while achieving comparable or better accuracy. In other scenarios (e.g., CNN-A-Adv-4 on CIFAR), Clip-and-Verify with BICCOS achieves 48.5% accuracy with a D.V. of only 843.47. This is the same or higher accuracy with a significantly smaller D.V. compared to standalone β -CROWN (46.5%, 2066.39 D.V.), GCP-CROWN with MIP cuts (48.5%, 3907.30 D.V.), and BICCOS (48.5%, 1319.56 D.V.). This illustrates that when Clip-and-Verify effectively prunes the search space, verification efficiency can improve even while tackling similarly challenging problems and achieving high accuracy.

A Breakdown Comparison between Verifiers Table 5 exhibits the instance-wise comparison on acasxu benchmark, as also illustrated in Figure 2a. Most of the instances in acasxu benchmark are easy to verify. On these instances, all the verifiers share similar verification time and D.V. . Instances

Table 5: Instance-wise breakdown comparison on acasxu benchmark between α , β -CROWN and Clip-and-Verify.

Method	Avg. on simpler instances		65		73	
	Time	D.V.	Time	D.V.	Time	D.V.
α , β -CROWN	1.0287	12615.35	timeout	-	19.1899	5474527
relaxed	1.0087	6759.11	9.1839	1876495	2.7422	291855
relaxed + reorder	1.0283	6467.96	7.6514	1416479	2.6385	229755
complete	1.0991	6350.11	6.8963	531381	3.5677	112431

Table 6: Ablation Studies on Verified accuracy (Ver.%), avg. per-verified-example domain visited number (D.V.) analysis for all method verified instances on different Clip-and-Verify components.

Dataset	Model	β -CROWN		GCP-CROWN with MIP cuts		BICCOS		Clip-and-Verify with β -CROWN		Clip-and-Verify with MIP cuts		Clip-and-Verify with BICCOS		Upper bound
		Ver.%	D.V.	Ver.%	D.V.	Ver.%	D.V.	Ver.%	D.V.	Ver.%	D.V.	Ver.%	D.V.	
MNIST	$\epsilon = 0.3$ and $\epsilon = 2/255$	71.0	2712.72	71.5	4447.54	76.0	3081.50	74.0	2395.96	73.5	1495.07	76.0	2636.99	76.5
	CNN-A-Adv	45.5	12621.38	48.5	8186.11	48.5	3622.71	45.5	1037.36	48.5	3704.86	48.5	2073.58	50.0
CIFAR	CNN-A-Adv-4	46.5	2066.39	48.5	3907.30	48.5	1319.56	46.5	298.89	48.5	2396.36	48.5	843.47	49.5
	CNN-A-Mix	42.0	6108.57	47.5	17609.84	48.0	8015.12	43.0	4462.48	47.5	9836.06	48.0	4189.92	53.0
	CNN-A-Mix-4	51.0	482.43	55.0	8922.50	56.0	3319.90	51.0	150.46	55.0	4304.32	56.5	3501.06	57.5
	CNN-B-Adv	47.0	7255.68	49.5	9846.32	51.0	5758.00	49	4951.07	51.5	6979.27	51.5	2677.94	65.0
	CNN-B-Adv-4	55.0	1776.66	58.5	4688.22	59.5	2711.15	56.5	649.92	60.0	3565.74	60.5	1095.30	63.5
cifar10-resnet	oval22	83.33	2105.76	87.5	7091.30	87.5	5428.60	86.11	478.0	88.89	2545.28	88.89	2643.15	100.0
	cifar100-2024	66.66	30949.95	83.33	16614.72	83.33	12730.08	73.33	20191.27	90.00	18891.25	90.00	14032.81	96.67
tinyimagenet-2024		59.5	1535.60	-	-	60.5	769.87	63.0	152.96	-	-	65.5	122.00	84.0
		67.5	830.92	-	-	69.0	497.52	70.0	188.02	-	-	72.0	118.34	78.5
vision-transformer 2024 [53]		59.0	149.75	-	-	-	-	61.0	84.84	-	-	-	-	100.0

73 and 65 stand out as they’re significantly harder to verify. For instance 73, clipping is able to cut down over 94% D.V. and over 80% verification time. While α , β -CROWN is unable to verify instance 65 within given timeout, Clip-and-Verify is able to verify it within 10 seconds. Cactus plots Figures 5 and 6 are also able to give a instance-wise comparison. Please see the following paragraph for a more detailed explanation.

Interpretation of the Cactus Plot Figures 5 and 6 illustrate the number of instances verified as runtime varies. Each line represents a method, with the x-axis showing the cumulative verified instances under a given timeout (y-axis). This style, used in VNN-COMP reports captures the trade-off between subproblem difficulty and runtime performance. The curve’s right end indicates total solved instances—further right means more instances verified within the timeout. A flat initial curve fragment reflects many easy instances solved quickly. A curve below and to the right of another shows consistently faster solving. The x-axis ordering reflects instance difficulty, from easy (left) to hard (right). For example, on the tinyimagenet benchmark (Figure 5d), our Complete Clipping + BICCOS variant solves easy instances faster and solves more hard instances. Across these benchmarks, the proposed Clip-and-Verify framework demonstrates a favorable balance: it reduces the number of hard subproblems, leading to more instances being verified within moderate time thresholds. This suggests that although our clipping procedures incur additional overhead that may be noticeable in easy instances, the overall net gain is highlighted in its ability to solve more instances and hard instances with shorter verification times.

Overall, these results show that Clip-and-Verify’s framework of enhanced linear bounding and “clipping” robustly scales across varying network depths and adversarial training schemes. The additional integration of MIP cuts or BICCOS bounding routines consistently pushes verified accuracy closer to each benchmark’s upper bound while balancing verification time and domain exploration. We plot and analyze these ablation studies across all benchmarks in Appendix D.2, confirming that our method not only raises coverage (Ver.%) but also leverages clipping and cutting plane methods to verify some of the hardest instances encountered.

D.3 Detailed Comparison with LP Solvers

To validate our claim that coordinate ascent is more efficient than LP solvers for our task, we integrated LP solvers directly into our BaB algorithm 4 to solve the clipping optimization problem. We compared our method (“Clip (ours)”) against the Gurobi solver using dual simplex with varying

iteration limits. The experiment was run on the acasxu benchmark instance 65, and we report the average time and bound error over 30 million LP calls during the entire BaB process. We report only Gurobi's `optimize()` time, ignoring all Python overhead for problem creation, giving an advantage to the LP solver.

As shown in Table 7, our method is **740x faster** than a 1-iteration LP heuristic and **880x faster** than a 10-iteration LP, while achieving comparable accuracy (0.00085 vs. 0.0007 mean error). The LP solver only achieves near-zero error with 10+ iterations, at which point it is intractably slow. The high fixed cost of LP solvers (even for 1 iteration) comes from presolve and initial basis factorization routines, which are far more expensive than our simple $\mathcal{O}(n \log n)$ sort.

Table 7: Comparison of our coordinate ascent ("Clip") vs. Gurobi dual simplex with iteration limits on acasxu (instance 65). Time and error are averaged over 30M calls.

Method	Avg. Time per Call	avg. Bounds Error	std. Bounds Error
LP-simplex (1 iter)	2.08s	0.401	1.45
LP-simplex (10 iter)	2.47s	0.0007	0.0018
LP-simplex (20 iter)	2.49s	0.0	0.0
Clip (ours)	0.0028s	0.00085	0.0019
LP-full (ground truth)	2.50s	0.0	0.0

We also investigated modern GPU-based LP solvers, specifically Google's PDLP [3, 4], which uses a Primal-Dual Hybrid Gradient (PDHG) algorithm. As shown in Table 8, while PDLP achieves high accuracy when it converges, it is not suitable for our use case. PDHG does not maintain a feasible solution during iterations. Under strict iteration limits, it has a very high failure rate, returning a NOT_SOLVED status and thus **failing to provide a valid dual bound** for our sound verification procedure. In contrast, dual simplex (Gurobi) and our method always return a valid, feasible bound. PDLP is designed for single, massive LPs, whereas our framework requires solving millions of independent, small LPs in parallel, a setting where our specialized GPU solver excels.

Table 8: Comparison with Google's PDLP. Our method is dramatically faster and, critically, always returns a valid bound, unlike PDLP which frequently fails under iteration limits. B.E. means Bound Error.

Method	Avg. Time per Call	avg. B.E	std. B.E	Failure Rate (%)
PDLP (1 iter)	2.23s	2e-8	4e-8	81.31%
PDLP (10 iter)	4.21s	1.4e-8	2.3e-8	48.26%
PDLP (20 iter)	6.02s	5.5e-8	1e-8	38.38%
Clip (ours)	0.0028s	0.00085	0.0019	0.0%
Gurobi-full	2.50s	0.0	0.0	0.0%

D.4 Details of Heuristics

Neuron Selection Heuristic. The performance of Complete Clipping can be sensitive to the choice of which intermediate neurons to refine. To guide this selection, we employ a heuristic based on the Branch-and-Bound for Split Recommendation (BaBSR) score, originally designed to select which neuron to branch on during BaB [13]. Specifically, we use the intercept score from BaBSR, which estimates the potential impact of refining a neuron's bounds on the final output relaxation. The score for a neuron k is calculated as:

$$\text{score}_k = \frac{\max(0, -l_k) \cdot \max(0, u_k)}{u_k - l_k} \cdot \max(0, -\text{mean}_{\underline{A}_k})$$

where l_k and u_k are the neuron's pre-activation lower and upper bounds, and $\text{mean}_{\underline{A}_k}$ relates to the intercept of the neuron's linear lower bound. A higher score indicates a looser relaxation and a greater potential for improvement. For each layer, we compute this score for all unstable neurons and select the top- k neurons for refinement with Complete Clipping.

To validate this choice, we conducted an ablation study on the `tinyimagenet-2024` and `cifar100-2024` benchmarks, comparing BaBSR against three alternative heuristics: random selection, prioritizing neurons with the largest bound gap ($U - L$), and prioritizing neurons with the largest bound product ($-U \times L$). As shown in Tables 9 and 10, the BaBSR heuristic consistently achieves the best verification time and explores the fewest subdomains, especially for the balanced top-20 setting. This confirms that BaBSR makes more effective choices, leading to earlier pruning and a more efficient search.

Table 9: Average verification time (s) per instance for different top-k heuristics.

Benchmark	Top-k Setting	Random	$-U \times L$	U-L	BaBSR
tinyimagenet	Top-10 per layer	16.3	16.5	16.8	16.2
	Top-20 per layer	18.1	17.9	17.2	15.1
	All neurons	22.4	22.4	22.3	22.7
cifar100	Top-10 per layer	26.2	26.6	27.0	26.1
	Top-20 per layer	29.1	28.8	27.7	24.3
	All neurons	36.5	36.4	36.4	36.5

Table 10: Average number of visited domains per instance for different top-k heuristics.

Benchmark	Top-k Setting	Random	$-U \times L$	U-L	BaBSR
tinyimagenet	Top-10 per layer	485	453	420	388
	Top-20 per layer	381	370	365	342
	All neurons	278	278	278	278
cifar100	Top-10 per layer	907	847	785	726
	Top-20 per layer	712	692	682	640
	All neurons	521	521	521	521

Table 11: Ablation Study on Top-k Neuron Selection on `cifar_cnn_a_mix`

Top-k	# Verified	Avg. # Domain Visited	Avg. Time (s)
0 (reduce to β -CROWN)	84	6108.57	4.20
20	86	4462.47	4.24
50	86	4372.79	4.83
all	85 (1 timeout)	1881.77	8.23

Table 11 shows the ablation study on the top-k neuron selection heuristic on `cifar_cnn_a_mix` benchmark, reveals that strategically prioritizing neurons based on their FSB intercept scores significantly enhances verification efficiency. Employing a moderate k (specifically top-k 20 and 50) leads to the best outcomes, successfully verifying more properties (86) while substantially reducing both the number of domains visited (by up to 28% compared to no heuristic) and the overall time (by up to 19%). In contrast, not using the heuristic (top-k 0) results in a less efficient search, while applying it to all unstable neurons (top-k “all”) drastically cuts down visited domains but incurs a prohibitive time cost and a timeout, indicating that the overhead of processing too many prioritized neurons outweighs the benefits of more targeted branching. This demonstrates a crucial trade-off, with intermediate k values striking an optimal balance between guided search and computational overhead.

Constraint Importance Heuristic The constraint importance heuristic is to sort the constraints for better performance. The constraints corresponding to hyperplanes closer to the center of the current input domain might be more immediately relevant or impactful for tightening the bounds or for cutting off a significant portion of the current feasible region $\mathcal{X}$.

First, for the current input box domain $\mathcal{X}$, determine its centroid $\hat{x}$. We can calculate $g^\top \hat{x} + h - \sum_{i=1}^n |g_i| \epsilon_i > 0$ (there is no $x \in \mathcal{X}$ satisfy the constraint in (2)) to check the infeasibility and

Table 12: Ablation Study on Top-k Neuron Selection on lsnc

Ratio of selected neurons	Total time (s)	Domains visited
0/6	13.37	18,073,174
1/6	11.25	7,254,214
2/6	10.54	4,914,243
3/6	9.70	3,072,863
4/6	9.23	2,088,198
5/6	8.69	1,739,412
6/6	7.66	871,14

$\mathbf{g}^\top \hat{\mathbf{x}} + h + \sum_{i=1}^n |g_i| \epsilon_i \leq 0$ (for all $x \in \mathcal{X}$ satisfy the constraint in (2)) to remove redundancy. Second, for each available linear constraint $\mathbf{g}^\top \mathbf{x} + h = 0$, calculate the geometric distance from the centroid $\hat{\mathbf{x}}$ to this hyperplane:

$$d = \frac{|\mathbf{g}^\top \mathbf{x}_0 + h|}{\|\mathbf{g}\|_2}$$

Then, sort the constraints in ascending order based on these calculated distances to prioritize constraints that are closer to the centroid. In methods like Algorithm 1 (Complete Clipping), processing more impactful constraints earlier might lead to faster convergence or more significant bound improvements in the coordinate ascent. Such an important metric could also be relevant to Algorithm 3 (Relaxed Clipping).

We tested the bound tightness improvement from the heuristic on acasxu instance 65. Clip-and-Verify improves the intermediate bounds for 470,772,542 times during input BaB, and 3.962% of them can be further tightened with constraint importance heuristic. For all these further improved bounds, we computed the empirical quantiles of the relative improvements, shown in Table D.4.

Percentile of problems	Max	0.1th	1th	5th	10th	25th	50th	75th	90th
Bound Improvement	4242.6%	270.79%	3.07%	0.64%	0.29%	0.10%	0.03%	0.01%	0.002%

These results reveal that the head of the distribution contains substantial refinements, with the maximum observed improvement reaching up to 4000%. This indicates that the heuristic can produce tightened bounds, potentially leading to earlier branch pruning or faster convergence.

E Limitation and Broader Impacts

Limitation. Our framework’s effectiveness is influenced by the trade-off between the precision of clipping and its computational overhead. While **Complete Clipping** utilizes GPU acceleration, its scalability can be impacted when networks have very large hidden layers combined with a high number of unstable neurons and sparse constraints. The memory complexity scales as $\mathcal{O}(B \times N \times M)$, where B is the number of BaB subproblems, N is the number of neurons in the largest layer, and M is the number of linear constraints. To mitigate this, our top-k neuron selection heuristic (Appendix D.3) strategically prioritizes the most critical neurons, balancing precision and cost.

Conversely, while **Relaxed Clipping** is highly efficient with a complexity of $\mathcal{O}(n)$ per input dimension, its effectiveness can diminish in very high-dimensional input spaces due to the curse of dimensionality, where axis-aligned relaxations may become looser.

However, the ultimate scalability of our method is not fundamentally limited by input dimensionality or layer width, but rather by the intrinsic difficulty of the verification query, which is an NP-hard problem. As demonstrated by our experiments on neural network control systems 2 even small networks with low-dimensional inputs can pose immense verification challenges. Clip-and-Verify proved essential in solving these hard instances. The key insight is that our approach adapts computational effort based on problem characteristics rather than being constrained by absolute dimensional limits, ensuring **practical applicability** across diverse verification scenarios.

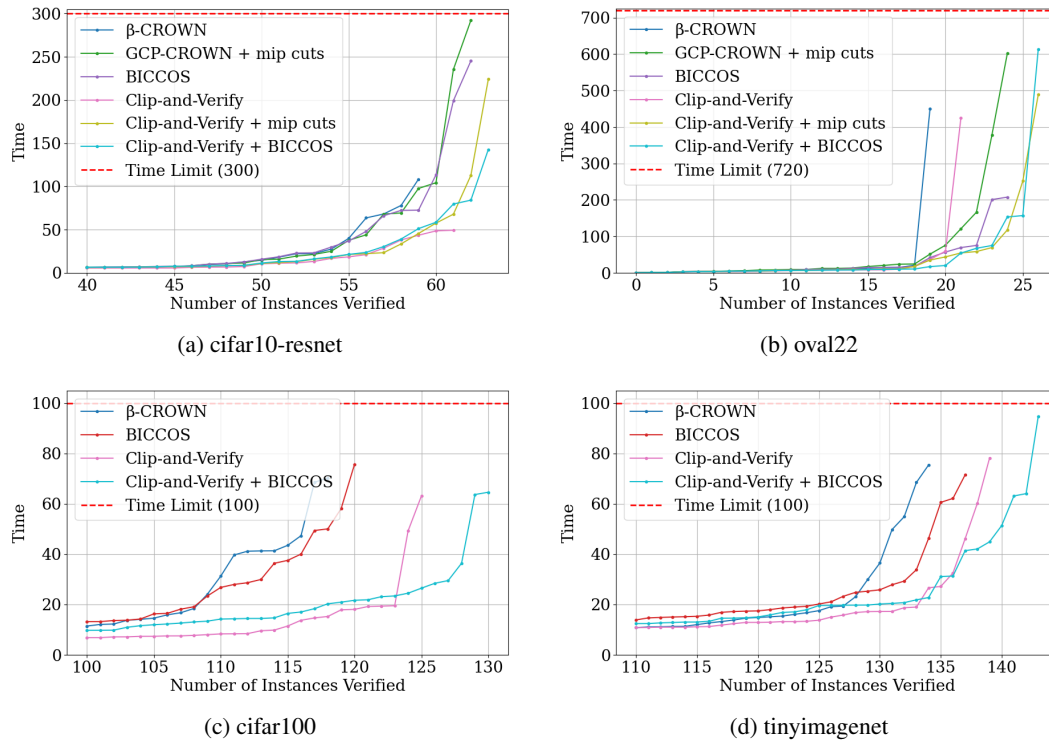
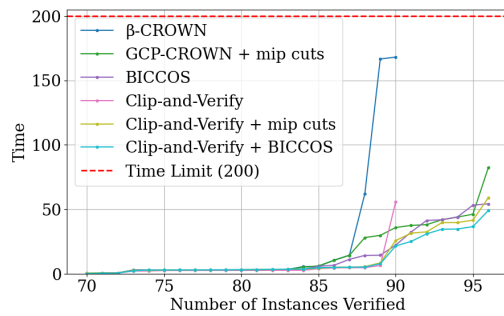
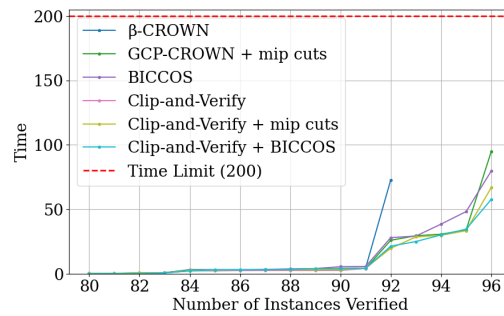


Figure 5: Plots for hard instances need to be solved by BaB in VNN-COMP benchmarks. It is important to note that in large-scale models with numerous properties to verify, the cutting plane method often incurs significant overhead. This is because our current approach processes all properties in batches, whereas the cutting plane method handles each property individually. Consequently, when addressing datasets like CIFAR-100 (99 properties) and TinyImageNet (199 properties), integrating with BICCOS introduces additional overhead.

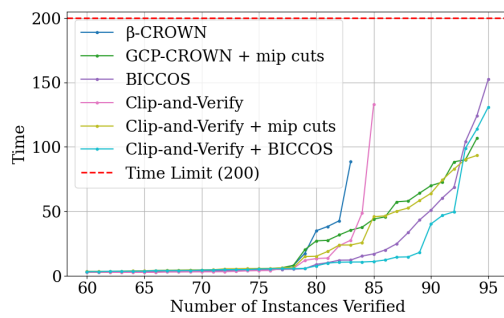
Broader Impacts Neural network verification is crucial for ensuring the safety and reliability of AI systems in critical applications such as autonomous vehicles, medical diagnosis, and financial trading. By significantly accelerating the verification process through efficient domain reduction, our work makes formal verification more practical for larger and more complex neural networks. This advancement enables broader adoption of verification techniques in real-world applications, potentially preventing catastrophic failures and building trust in AI systems. Clip-and-Verify’s integration with existing frameworks ensures immediate applicability, allowing organizations to implement stronger safety guarantees without substantial overhead. While this work strengthens the safety of AI systems, it is important to note that verification tools should be part of an approach to AI safety, including robust testing interpretability, and ethical guidelines.



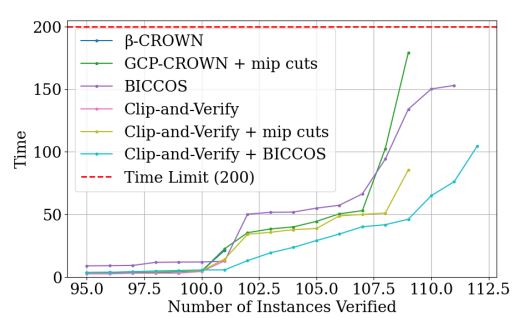
(a) cifar_cnn_a_adv



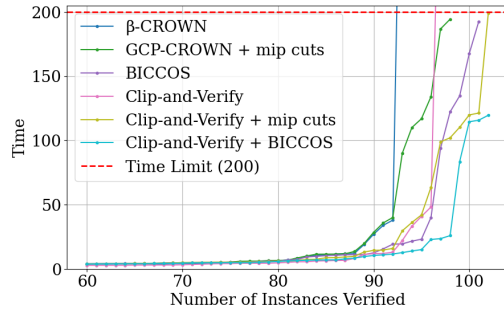
(b) cifar_cnn_a_adv4



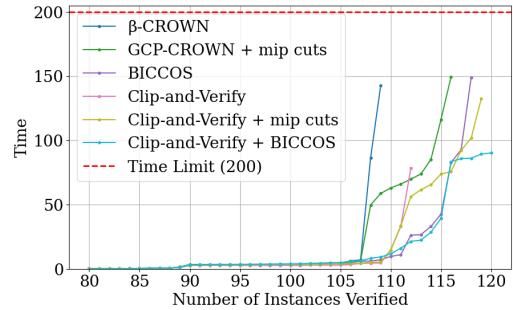
(c) cifar_cnn_a_mix



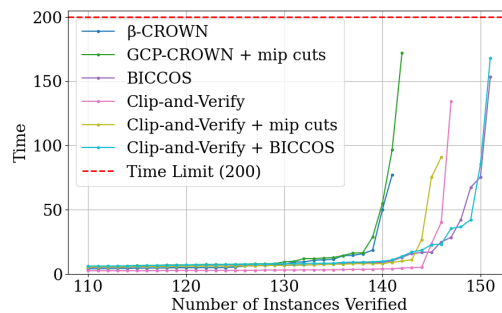
(d) cifar_cnn_a_mix4



(e) cifar_cnn_b_adv



(f) cifar_cnn_b_adv4



(g) mnist_cnn_a_adv

Figure 6: Plots for hard instances need to be solved by BaB in SDP benchmarks.

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: Our claims match our theoretical and empirical results. We present results across several benchmarks to demonstrate that they can generalize to other settings.

Guidelines:

- The answer NA 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 NA 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: Limitations have been explicitly discussed in Appendix E.

Guidelines:

- The answer NA 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: Theoretical results have been proven.

Guidelines:

- The answer NA 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: Algorithms have been clearly described with technical details discussed. Parameters for experiments have been included in the appendix. The code will be open-sourced.

Guidelines:

- The answer NA 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: Code will be open-sourced.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://nips.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://nips.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, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: Experimental details have been discussed in the Experiments section and also in the Appendix.

Guidelines:

- The answer NA 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: [NA]

Justification: Verification results are deterministic on the benchmarks, and no error bars need to be provided.

Guidelines:

- The answer NA 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: Time has been reported in tables and we only need one machine and one GPU to run all experiments. The specific hardware details are reported in the Appendix.

Guidelines:

- The answer NA 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: We have read and acknowledged the NeurIPS Code of Ethics.

Guidelines:

- The answer NA 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: Societal impacts have been discussed explicitly in Appendix E.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA 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., pretrained language models, image generators, or scraped datasets)?

Answer: [\[Yes\]](#)

Justification: No data or models are released.

Guidelines:

- The answer NA 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've cited the sources of the benchmarks we used, such as those from the VNN-COMP

Guidelines:

- The answer NA 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: [NA]

Justification: No new assets are introduced in this paper.

Guidelines:

- The answer NA 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: [NA]

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

Guidelines:

- The answer NA 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: [NA]

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

Guidelines:

- The answer NA 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: [NA]

Justification: The paper does not use LLMs for important, original, or non-standard component of the core methods in this research.

Guidelines:

- The answer NA 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 (<https://neurips.cc/Conferences/2025/LLM>) for what should or should not be described.