CLAD: CONSTRAINED ABSTRACT DOMAIN FOR NEURAL NETWORK VERIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural network verification (NNV) formally verifies that a network satisfies a specified property for all inputs within a defined region. Modern NNV tools employ *abstract domains* to compute a sound over-approximation of the network's behavior from the given input region, thus the tightness of these abstractions essentially determines efficiency. A long line of increasingly precise domains has been developed, but they all describe the valid input region in the same restrictive way, *e.g.*, an ℓ_p-norm ball. A practical input region is rarely a simple ℓ_p ball, but rather a combination ℓ_p ball with additional constraints. Verifying a network over such a region with existing abstraction produces a loose over-approximation, which results in either failing to verify a property or spurious counterexamples. We introduce *Constrained Lagrangian Abstract Domain* (CLAD), a new abstract domain that computes a sound over-approximation of neural networks over input regions defined by a combination of convex constraints. CLAD propagates these constraints and tightens bounds over the true feasible region. However, bounding a neuron over the intersection of these constraints has no closed-form solution, so CLAD relaxes each constraint into the objective with a Lagrange multiplier and solves the resulting max-min problem with a projected primal-dual method, alternating a projected gradient step on the input with a multiplier update. CLAD supports any convex constraint with a subgradient, *e.g.*, from automatic differentiation. We evaluate CLAD on 1,944 instances across four convolutional networks with motion-blur structured perturbations with halfspace or ℓ_2-ball constraints. On standard unconstrained ℓ_∞ property, CLAD verifies as many instances as GCP-CROWN at a similar runtime. On constrained properties, CLAD verifies 60% more instances than GCP-CROWN on ℓ_2-ball properties, and 22% more in total.

1 INTRODUCTION

Deep neural networks (DNNs) have been increasingly adopted in real-world applications and safety-critical domains such as autonomous driving Shao et al. (2023) and medical diagnostics Bizjak et al. (2022). However, they can fail in unexpected ways, *e.g.*, physical obstructions like stickers on road signs can fool traffic classifiers Eykholt et al. (2018). *Neural network verification* (NNV), a new fledgling field (Wang et al., 2018; Singh et al., 2019; Katz et al., 2022; Wang et al., 2021) aims to formally verify that a network behaves correctly under adverse conditions (Huang et al., 2020), *e.g.*, a network satisfies a specified property for all possible inputs within a defined region. In recent years, researchers have developed many NNV algorithms and tools (Chiu et al., 2025; Zhou et al., 2024; Duong et al., 2025a;b; Duong & Nguyen, 2025; Duong et al., 2025c; 2026a;b; Wu et al., 2024; PyRAT, 2024). State-of-the-art (SoTA) NNV tools, which adapt program analysis and constraint solving techniques, have been shown to be effective and scale to large networks with millions of parameters, bringing many excitements and promises to NNV research Kaulen et al. (2025).

Despite recent advances, scalability remains a major challenge in NNV. Modern NNV tools adopt *branch-and-bound* (BaB) Bunel et al. (2020b), which recursively divides a verification problem into subproblems, and at each subproblem, used an *abstract domain*, inspired by abstract interpretation (Cousot & Cousot, 1977), to compute an over-approximation of the network's behavior from given input region. *The tightness of abstraction is what ultimately determines the verification efficiency*, *e.g.*, a more precise domain prunes more subproblems before it explodes combinatorially.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

A long line of NNV work has therefore pursued computationally efficient yet precise abstract domains, ranging from fast but coarse intervals (IBP (Wang et al., 2018)) to relational domains such as ZONOTOPE (Singh et al., 2018) and the polytope-based domains, *e.g.*, CROWN (Zhang et al., 2018) and DEEPPOLY (Singh et al., 2019). More recent refinements push this precision further, including optimizing relaxations (α-CROWN (Xu et al., 2020b)), tightening ℓ_2-ball (SDP-CROWN (Chiu et al., 2025)), or adding cutting planes on hidden neurons (GCP-CROWN (Zhang et al., 2022b)). However, these works are all confined to compute abstraction of one ℓ_p-norm ball properties.

Recent work has moved toward richer, more realistic properties, such as structural robustness against coordinated input transformations, *e.g.*, filtering (Duong et al., 2026a) or combined perturbations (Duong et al., 2026b). However, these efforts do not enrich the abstract domain itself but *reduce* the property to a standard ℓ_∞ problem by prepending an auxiliary subnetwork that generates the perturbation from fewer control variables. This thus restricts the property to those that can be reduced by changing the architecture of the network and cannot express arbitrary input constraints.

The need for an abstract domain that can directly handle more expressive input regions is not hypothetical: whenever an input region is described *accurately*, it is rarely a single ℓ_p ball. A growing line of work that reasons directly about the *input space* of a network such as *verifiable input space* - the largest region on which a network's outputs are certified safe - characterize it as a union of balls bounded by additional safety constraints (Chehade et al., 2025). Likewise, methods that compute the *preimage* of a target output set represent it as a union and intersection of many halfspaces (Zhang et al., 2024). In both cases, the region of interest is expressed as a base ℓ_p ball combined with additional convex constraints.

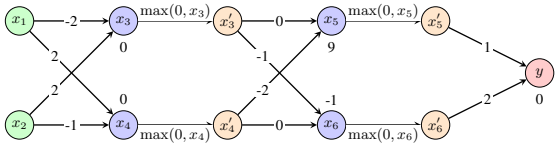
In this work, we propose *Constrained Lagrangian Abstract Domain* (CLAD), a new abstract domain that verifies DNNs over input regions defined by a combination of convex constraints. CLAD propagates these constraints through the network and tightens every neuron's bounds over the true feasible region, *e.g.*, the combination of all the input constraints. Intuitively, each added constraint rules out inputs the network can never receive, so CLAD no longer has to account for them and can produce a tighter bound. To tackle the challenge of tightening the bounds over combined convex constraints (which has no closed form), CLAD relaxes each constraint into the objective with a Lagrange multiplier and solves the resulting max-min problem with a projected primal-dual method, alternating a projected gradient step on the input with a multiplier update. Each constraint is supplied as a convex constraint function with a subgradient, so CLAD supports any such constraint, *e.g.*, the intersection of a halfspace and ℓ_2 balls, through auto-gradient. The resulting bound is *sound* and *tight*.

We evaluate CLAD on 1,944 instances across four convolutional networks with motion-blur structured perturbations, each augmented with halfspace or ℓ_2-ball constraints. On the standard unconstrained ℓ_∞ property, CLAD verifies as many instances as GCP-CROWN at a similar runtime. On constrained properties the gap widens sharply: CLAD verifies about 60% more instances than GCP-CROWN on ℓ_2-ball properties, at a higher runtime, because every baseline bounds over the enclosing ℓ_∞ interval, while CLAD propagates the constraint and tightens each neuron bound over the feasible region. Overall, CLAD verifies 22% more instances than the SoTA GCP-CROWN and $3.3\times$ as many on the largest network, and its advantage grows as input perturbation strength increases.

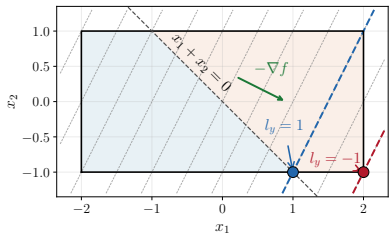
Our contributions include: (1) A new abstract domain CLAD that handles additional convex constraints by relaxing each one into the objective with a Lagrange multiplier and computing its gradient by automatic differentiation; (2) We prove CLAD returns a sound bound at every iteration, so it can stop early, and mechanize its duality results in Lean; (3) We establish the stopping conditions on the constraints under which CLAD returns a tight bound; and (4) We evaluate CLAD on 1,944 instances with and without halfspace and ℓ_2-ball constraints, where it verifies 22% more instances than GCP-CROWN overall and about 60% more on ℓ_2-ball properties, while matching GCP-CROWN on unconstrained ℓ_∞ properties at a similar runtime.

2 MOTIVATING EXAMPLE

We illustrate CLAD using a simple DNN with ReLU in Fig. 1a. Learned weights are shown on the edges, while learned bias for each neuron is shown above or below it. For illustration, we use x and x' neurons to represent the pre-and post-activations of the ReLU layers, *e.g.*, x'_3 and x'_4 are the



(a) A simple neural network with ReLU. x_1, x_2 are input and y is output.



(b) Concretizing $y \geq -2x_1 + x_2 + 4$. Without $\mathcal{H}$ (red): returns $l_y = -1$ at infeasible $(2, -1)$. CLAD (blue): returns $l_y = 1$ at feasible $(1, -1)$.

Fig. 1: Motivating example: network architecture (a) and constrained concretization (b).

post-activation values after applying ReLU to x_3 and x_4, respectively. The computation is then:

$$x_3 = -2x_1 + 2x_2, \quad x_3' = \text{ReLU}(x_3), \quad x_4 = 2x_1 - x_2, \quad x_4' = \text{ReLU}(x_4), \quad y = x_5' + 2x_6'$$
$$x_5 = -2x_4' + 9, \quad x_5' = \text{ReLU}(x_5), \quad x_6 = -x_3' - 1, \quad x_6' = \text{ReLU}(x_6) \tag{1}$$

Suppose we want to verify that y is always positive for all inputs (x_1, x_2) in a certain input region:

$$\forall (x_1, x_2), \quad \underbrace{x_1 \in [-2, 2] \wedge x_2 \in [-1, 1]}_{\mathcal{B}} \wedge \underbrace{x_1 + x_2 \leq 0}_{\mathcal{H}} \implies y > 0 \tag{2}$$

This property involves the interval range $\mathcal{B} : x_1 \in [-2, 2]$ and $x_2 \in [-1, 1]$ and the half-space[1] constraint $\mathcal{H} : x_1 + x_2 \leq 0$.

To handle complex constraints, CLAD associates each variable v_i with symbolic polyhedral bounds (to preserve relational information) and concrete scalar bounds l_i, u_i (to check ReLU stability). The key difference from prior work is *how* these scalars are computed: while DEEPPOLY Singh et al. (2019) evaluates the symbolic expressions merely over the input box $\mathcal{B}$, CLAD yields tighter scalars by optimizing over the full feasible set $\mathcal{B} \cap \mathcal{H}$ (where $\mathcal{H} : x_1 + x_2 \leq 0$ is the halfspace constraint).

Starting with inputs $x_1 \in [-2, 2]$ and $x_2 \in [-1, 1]$, the first affine transformer computes:

$$x_3 = -2x_1 + 2x_2, \qquad x_4 = 2x_1 - x_2 \tag{3}$$

Using the input bounds, we get concrete ranges $x_3 \in [-6, 6]$ and $x_4 \in [-5, 5]$. Since these ranges cross zero, both neurons are unstable. CLAD therefore applies the standard linear relaxation (§A.2). For the lower bound we take $x' \geq 0$, and for the upper bound we follow DEEPPOLY and compute:

$$0 \leq x_3' \leq 0.5\, x_3 + 3, \qquad 0 \leq x_4' \leq 0.5\, x_4 + 2.5 \tag{4}$$

Propagating these abstractions through the subsequent layers yields the final output y. By back-substituting the intermediate symbolic bounds layer-by-layer, CLAD expresses the lower bound of y in terms of the original inputs (x_1, x_2):

$$y \geq -2x_1 + x_2 + 4 \tag{5}$$

To verify $y > 0$, CLAD computes the concrete lower bound of Eq. 5 over the feasible input region:

$$l_y \;=\; \min_{x \in \mathcal{B}} f(x) \;=\; \min_{x \in \mathcal{B}} \left(-2x_1 + x_2 + 4\right) \quad \text{subject to} \quad x_1 + x_2 \leq 0 \tag{6}$$

To solve Eq. 6, we use a Lagrangian technique (Boyd & Vandenberghe, 2004; Dvijotham et al., 2018), which converts the constrained optimization into an unconstrained one by folding the constraints into the objective as penalty terms that discourage infeasible solutions. Let $\mathcal{L}(x, \mu)$ denote the Lagrangian, which contains the minimized objective f and the constraint function $h(x) = x_1 + x_2 \leq 0$ weighted by $\mu \geq 0$:

$$\mathcal{L}(x, \mu) \;=\; (-2x_1 + x_2 + 4) \;+\; \mu\,(x_1 + x_2), \qquad \mu \geq 0 \tag{7}$$

[1] A linear constraint in 2D involving two variables.

where $\mu \geq 0$ controls how strongly violations are penalized. For every feasible x, the term $\mu(x_1 + x_2)$ is non-positive, so $\mathcal{L}(x, \mu) \leq f(x)$ and hence $\min_{x \in \mathcal{B}} \mathcal{L}(x, \mu) \leq l_y$ for every $\mu \geq 0$. We now solve the constrained problem Eq. 6 using the Lagrangian Eq. 7.

In particular, if μ is too small (*e.g.*, $\mu = 0$), the penalty barely matters and the minimizer runs to infeasible point $(2, -1)$ (unconstrained solution). If μ is too large, the term $\mu(x_1 + x_2)$ becomes a huge *reward* for points deep inside the halfspace (where $x_1 + x_2$ is very negative). The best μ sits in between, balancing the two effects. CLAD searches for μ that gives the largest, and thus tightest, lower bound via an iterative loop that raises μ proportionally to current violation until the penalty is strong enough to keep $(x_1, x_2) \in \mathcal{H}$:

$$\max_{\mu \geq 0} \min_{x \in \mathcal{B}} \mathcal{L}(x, \mu) \;=\; \max_{\mu \geq 0} \min_{x \in \mathcal{B}} \left\{ (-2x_1 + x_2 + 4) \;+\; \mu\,(x_1 + x_2) \right\} \tag{8}$$

CLAD solves this dual optimization problem by alternating between two steps: (i) adjusting x to decrease the Lagrangian and (ii) adjusting μ to increase the penalty on violations until convergence. Each iteration performs two steps:

(1) Primal step. CLAD moves (x_1, x_2) along the negative gradient of $\mathcal{L}$, which decreases the objective $-2x_1 + x_2 + 4$. The penalty $\mu(x_1 + x_2)$ adds $-\mu(1, 1)$ to this direction at every iterate so it lowers $x_1 + x_2$ and pulls a violating point back toward $\mathcal{H}$ with a strength set by μ. After the move, (x_1, x_2) is clamped back into the interval $\mathcal{B}$

(2) Dual step. CLAD increases the penalty weight μ by an amount proportional to how much $\mathcal{H}$ is currently violated. This makes subsequent primal steps push more strongly toward the feasible region, so the two steps drive (x_1, x_2) toward a point that is both low-objective and feasible.

The solver stops once its certified bound stops improving, returning $l_y = 1$, attained at $(1, -1)$ on the boundary $x_1 + x_2 = 0$ instead of $l_y = -1$ at infeasible $(2, -1)$.

Result. Finally, we check the result against the specification, which confirms the computed lower bound $l_y = 1 > 0$ successfully verifies the specification $y > 0$ over the feasible input region $\mathcal{B} \cap \mathcal{H}$, where ignoring $\mathcal{H}$ gives the loose bound $l_y = -1$.

3 THE CLAD ABSTRACTION DOMAIN

Alg. 1 shows how CLAD computes output bounds for a DNN $\mathcal{N}$ over an input ball $\mathcal{B}$ with constraints $\{h_j\}_{j=1}^{M}$. CLAD first traverses $\mathcal{N}$ in topological order and computes concrete bounds at activation nodes to construct their linear relaxations (line 2), while linear nodes are propagated exactly in symbolic form. For each node v_i, the abstraction maintains symbolic and concrete bounds:

$$\underline{A}_i x + \underline{b}_i \leq v_i \leq \overline{A}_i x + \overline{b}_i, \qquad l_i \leq v_i \leq u_i.$$

The symbolic bounds preserve the dependence on the inputs, while the concrete bounds determine the activation relaxation. To bound each activation v_i, CLAD collects its predecessors (line 3) and initializes the symbolic bounds (line 4). It then traverses the subgraph in reversed topological order, tracing v_i backward until both symbolic bounds are expressed in the inputs (line 5).

During the traversal, CLAD substitutes the exact affine map of each linear node (line 11). For a linear or convolution node with weight W_k and bias c_k, `Substitute` composes the exact affine:

$$\texttt{Substitute}(A, b, v_k) = (AW_k,\ Ac_k + b).$$

At each activation, CLAD constructs lower and upper linear relaxations (§A.2) from its concrete bounds $[l_k, u_k]$ (line 7). It selects which relaxation to substitute based on the coefficient sign (line 8-line 9). For $r(x) = d_r \odot x + e_r$ and $r'(x) = d_{r'} \odot x + e_{r'}$, the substitution is

$$\texttt{Substitute}(A, b, r, r') = \left(A^{+}\mathrm{diag}(d_r) + A^{-}\mathrm{diag}(d_{r'}),\ A^{+}e_r + A^{-}e_{r'} + b\right) \tag{9}$$

where $A^{+} = \max(A, 0)$ and $A^{-} = \min(A, 0)$ elementwise. Passing $(\underline{r}_k, \overline{r}_k)$ gives a sound lower bound (a positive coefficient preserves the bound direction), while passing $(\overline{r}_k, \underline{r}_k)$ gives a sound upper bound (a negative coefficient reverses the direction).

Finally, CLAD concretizes the symbolic bounds into $[l_i, u_i]$ by optimizing them over the feasible input set (line 13 -line 14). This feasible set combines the input ball $\mathcal{B}$ with the additional constraints $\{h_j\}_{j=1}^{M}$ (§3.1). CLAD solves this optimization with the projected primal dual method (§3.2 and Alg. 2). CLAD uses the resulting $[l_i, u_i]$ to construct later activation relaxations and returns the output range as the final certificate.

Algorithm 1: CLAD algorithm.

input: DNN $\mathcal{N}$; input ball $\mathcal{B} = \{x : \|x - x_0\|_p \le \varepsilon\}$; constraint functions $\{h_j\}_{j=1}^{M}$
output: output bounds $[l_{\text{out}}, u_{\text{out}}]$

for v_i *in* `TopoSort`$(\mathcal{N})$ **do**
  if `IsActivation`$(v_i) \vee v_i = v_{\text{out}}$ **then**
    $\mathcal{G}_i \leftarrow \{v_i\} \cup$ `Predecessors`(v_i) ; `// sub-DAG feeding` v_i
    $(\underline{A}_i, \underline{b}_i), (\overline{A}_i, \overline{b}_i) \leftarrow (I, 0), (I, 0)$; `// init to the trivial bound` $v_i \le v_i \le v_i$
    `// compute` v_i `symbolically in the inputs` x
    for v_k *in* `ReverseTopoSort`$(\mathcal{G}_i)$ **do** `// back-substitute from` v_i `to input` x
      if `IsActivation`(v_k) **then**
        $(\underline{r}_k, \overline{r}_k) \leftarrow$ `Relax`(v_k, l_k, u_k) ; `// linear bound relaxation` $\underline{r}_k \le v_k \le \overline{r}_k$
        $(\underline{A}_i, \underline{b}_i) \leftarrow$ `Substitute`$(\underline{A}_i, \underline{b}_i, \underline{r}_k, \overline{r}_k)$; `// substitute lower bound`
        $(\overline{A}_i, \overline{b}_i) \leftarrow$ `Substitute`$(\overline{A}_i, \overline{b}_i, \overline{r}_k, \underline{r}_k)$; `// substitute upper bound`
      else
        $(\underline{A}_i, \underline{b}_i) \leftarrow$ `Substitute`$(\underline{A}_i, \underline{b}_i, v_k)$; `// substitute lower bound`
        $(\overline{A}_i, \overline{b}_i) \leftarrow$ `Substitute`$(\overline{A}_i, \overline{b}_i, v_k)$; `// substitute upper bound`
    `// concretize each side over` $\mathcal{B} \cap \bigcap_j \{h_j \le 0\}$ `(Alg. 2)`
    $l_i \leftarrow$ `Concretize`$(\underline{A}_i, \underline{b}_i, \mathcal{B}, \{h_j\}, +1)$; `// compute concrete lower bound`
    $u_i \leftarrow$ `Concretize`$(\overline{A}_i, \overline{b}_i, \mathcal{B}, \{h_j\}, -1)$; `// compute concrete upper bound`
return $[l_{\text{out}}, u_{\text{out}}]$;

3.1 CONSTRAINT FORMULATION

The input property of a NNV instance can contain one or more constraints. Adding a constraint to CLAD requires only a *constraint function* that measures its violation.

Constraint. Each constraint has the standard form $h_j(x) \le 0$, where $h_j : \mathbb{R}^D \to \mathbb{R}$. The $h_j(x)$ is nonpositive iff x satisfies the constraint; otherwise, its positive value measures the violation.

Penalty. The nonnegative penalty associated with h_j is $\phi_j(x) = \max\big(0, h_j(x)\big)$, which equals zero when the constraint is satisfied and measures the violation otherwise.

Example. For the ℓ_2 ball $\mathcal{B}_2 = \{x : \|x - x_c\|_2 \le r\}$, the constraint function is $h(x) = \|x - x_c\|_2 - r$ and the penalty is $\phi(x) = \mathrm{ReLU}\big(\|x - x_c\|_2 - r\big)$. CLAD uses ϕ_j in place of h_j: it is convex, zero on the feasible set, and has subgradient 0 at its kink, so the non-differentiable center x_c never enters.

3.2 CONTRAINED ABSTRACTION

Linear relaxation methods (Zhang et al., 2018; Singh et al., 2019) bound each neuron v_i in the inputs x by $\underline{A}_i x + \underline{b}_i \le v_i \le \overline{A}_i x + \overline{b}_i$. *Concretization* optimizes these expressions over all valid inputs to obtain $[l_i, u_i]$. These scalars determine the stability of an activation and its relaxation. A piecewise linear activation is *stable* if $[l_i, u_i]$ lies within one linear piece, where the activation is linear and needs no approximation. It is *unstable* if $[l_i, u_i]$ spans the nonlinear region, requiring a linear relaxation. Thus, tighter ranges keep more neurons stable and yield tighter bounds. For the ℓ_p ball $\mathcal{B} = \{x : \|x - x_0\|_p \le \varepsilon\}$, Hölder's inequality gives the closed form:

$$u_i \;=\; \overline{A}_i x_0 + \varepsilon \,\|\overline{A}_i\|_q + \overline{b}_i \qquad l_i \;=\; \underline{A}_i x_0 - \varepsilon \,\|\underline{A}_i\|_q + \underline{b}_i \tag{10}$$

where $1/p + 1/q = 1$. When $p = \infty$ and $q = 1$, the worst case independently moves each input coordinate to the interval boundary matching the sign of its coefficient in $\overline{A}_i$. The result is its value at x_0 plus ε times the sum of absolute coefficients, or the ℓ_1 norm. An additional constraint such as $x_1 + x_2 \le 0$ may remove the interval boundaries from feasible space, thus, closed-form solutions from Hölder is no longer valid. CLAD concretizes each neuron over the feasible space $\mathcal{B} \cap \mathcal{H}$:

$$l_i \;=\; \min_{x \in \mathcal{B} \cap \mathcal{H}} \underline{A}_i x + \underline{b}_i \qquad -u_i \;=\; \min_{x \in \mathcal{B} \cap \mathcal{H}} -\big(\overline{A}_i x + \overline{b}_i\big) \tag{11}$$

where $\mathcal{B} = \{x : \|x - x_0\|_p \le \varepsilon\}$ be the base ℓ_p norm ball input property and $\mathcal{H} = \{x : h_j(x) \le 0,\ j = 1, \ldots, M\}$ the feasible set formed by the base interval and additional constraints.

There are three common methods to solve the constrained concretization problem (Boyd & Vandenberghe, 2004): (i) *Dual ascent* evaluates $\mathrm{d}(\mu)$ exactly and ascends along its supergradient, but general h_j does not provide the closed form inner minimization; (ii) *Primal dual interior point methods* are accurate but solve each instance with costly Newton steps, so they do not scale to CLAD concretization; and (iii) *Projected primal dual methods* (Arrow et al., 1958; Nedić & Ozdaglar, 2009) use a first order descent step in x projected onto $\mathcal{B}$ and an ascent step in μ projected onto $\{\mu \geq 0\}$.

Therefore, CLAD uses projected primal-dual method to solve Eq. 11. We present the procedure for a sound and tight lower bound l_i and write its back-substituted bound as $Ax + b$; the upper bound is symmetric and minimizes $-(\overline{A}_i x + \overline{b}_i)$, as in Eq. 11.

Lagrangian. The Lagrangian (Boyd & Vandenberghe, 2004; Dvijotham et al., 2018) handles constrained optimization by replacing constraints $h_j(x) \leq 0$ with a weighted sum of their values in the objective. Each constraint has a nonnegative μ_j. Concretely, the Lagrangian assigns a dual multiplier $\mu_j \geq 0$ to each inequality constraint $h_j(x) \leq 0$ in Eq. 11 and adds them into the objective:

$$\mathcal{L}(x, \mu) \;=\; Ax + b \;+\; \sum_{j=1}^{M} \mu_j \, h_j(x) \tag{12}$$

For every feasible $x \in \mathcal{B} \cap \mathcal{H}$, $h_j(x) \leq 0$ and $\mu_j \geq 0$, so each term $\mu_j h_j(x) \leq 0$. Thus, $\mathcal{L}(x, \mu) \leq Ax + b$, with equality when all constraints are active.

Dual problem. The dual function minimizes the Lagrangian over $\mathcal{B}$: $\mathrm{d}(\mu) = \min_{x \in \mathcal{B}} \mathcal{L}(x, \mu)$. Substituting Eq. 12 gives the dual problem

$$\max_{\mu \geq 0} \mathrm{d}(\mu) = \max_{\mu \geq 0} \min_{x \in \mathcal{B}} \left(Ax + b + \sum_{j=1}^{M} \mu_j h_j(x) \right) \tag{13}$$

Every $\mu \geq 0$ gives a sound lower bound on l_i (Thm. C.1). Thus, if the DNN verifier stops solving $\max_{\mu \geq 0} \mathrm{d}(\mu)$ due to a time constraint, its current nonoptimal μ still gives a valid bound $\mathrm{d}(\mu)$. For $x_\mu \in \arg\min_{x \in \mathcal{B}} \mathcal{L}(x, \mu)$, the constraint values form a supergradient of the concave dual function:

$$\mathrm{d}(\mu') \leq \mathrm{d}(\mu) + \sum_{j=1}^{M} h_j(x_\mu)(\mu'_j - \mu_j) \tag{14}$$

where $\mu' \geq 0$ is other multiplier vector. Thus, a positive $h_j(x_\mu)$ raises μ_j and vice versa.

Primal-dual solver. Alg. 2 handles both bounds with a sign s: it minimizes $s(Ax + b)$ over $\mathcal{B} \cap \mathcal{H}$, with $s = +1$ for the lower bound and $s = -1$ for the upper bound, whose result it negates. Its Lagrangian is $\mathcal{L}_s(x, \mu) = s(Ax + b) + \sum_j \mu_j h_j(x)$, which equals Eq. 12 when $s = +1$. Each iteration of Alg. 2 performs two steps. The *primal step* computes the gradient $q = s \cdot A + \sum_j \mu_j \nabla h_j(x)$ of $\mathcal{L}_s$ where each gradient ∇h_j is obtained via automatic differentiation and descends as $x \leftarrow x - \eta q$. The *dual step* updates each multiplier as $\mu_j \leftarrow \max(0,\, \mu_j + \tau\, h_j(x))$, increasing μ_j when constraint h_j is violated and decreasing it when satisfied, so the weight on each violated constraint grows until x is feasible. Concretely, the two steps update:

$$\begin{aligned} q_t &= sA + \sum_{j=1}^{M} \mu_{j,t} \nabla h_j(x_t) \qquad & \widetilde{x}_{t+1} &= x_t - \eta_t q_t \\ \mu_{j,t+1} &= \max\big(0, \mu_{j,t} + \tau h_j(\widetilde{x}_{t+1})\big) \quad & x_{t+1} &= \Pi_{\mathcal{B}}(\widetilde{x}_{t+1}) \end{aligned} \tag{15}$$

The primal variable is initialized at the closed-form Hölder minimizer of $s \cdot Ax$ over $\mathcal{B}$ and projected back onto $\mathcal{B}$ by $\Pi_{\mathcal{B}}$ after each step. Hölder's inequality gives the warm start value

$$\min_{x \in \mathcal{B}} s(Ax + b) = s(Ax_0 + b) - \varepsilon \|A\|_q, \qquad \frac{1}{p} + \frac{1}{q} = 1 \tag{16}$$

For $p = \infty$, a minimizer is $x^{(0)} = x_0 - \varepsilon \operatorname{sign}(sA)$. The primal variable is not projected onto $\mathcal{H}$ because the additional constraints control x via the regularization terms in the Lagrangian. The projection used by the primal step is

$$\Pi_{\mathcal{B}}(z) = \arg\min_{x \in \mathcal{B}} \|x - z\|_2 \tag{17}$$

Algorithm 2: CLAD Concretization Algorithm.

input: weight A and bias b of a linear bound; ℓ_p ball $\mathcal{B} = \{x : \|x - x_0\|_p \leq \varepsilon\}$; constraint functions $\{h_j\}_{j=1}^{M}$; sign $s \in \{-1, +1\}$

hyper-params: primal step size η, dual step size τ, initial multiplier μ_0, tolerance ϵ, patience K, number of iterations T

output: lower bound if $s = +1$, upper bound if $s = -1$

1 $x \leftarrow \arg\min_{x \in \mathcal{B}} s \cdot Ax$; $\beta \leftarrow \min_{x \in \mathcal{B}} s(Ax + b)$; // warm start and μ=0 bound, Eq. 16

2 $\mu_j \leftarrow \mu_0 \quad \forall j \in \{1, \ldots, M\}$; // initialize dual multipliers

3 **for** $t = 1$ **to** T **do** // run optimization for T iterations

4 $q, x' \leftarrow \texttt{PrimalStep}(x, \mu, A, \{h_j\}, \eta)$ // primal descent

5 $dx \leftarrow x' - x$; $y \leftarrow \big(s \cdot A + \sum_j \mu_j \nabla h_j(x')\big) - q$; // changes in x and gradient

6 $x \leftarrow x'$; // update primal variable

7 $\mu \leftarrow \texttt{DualStep}(x, \mu, \{h_j\}, \tau)$ // dual ascent

8 $\beta \leftarrow \max\big(\beta,\ \hat{\mathrm{d}}(x, \mu)\big)$; // certified bound, Eq. 20

9 **if** $\|y\|_2^2 > \epsilon$ **then** $\eta \leftarrow \max\big(0,\ \min\big(|dx^\top y| \,/\, \|y\|_2^2,\ 1\big)\big)$; // Barzilai-Borwein step size

10 $x \leftarrow \Pi_{\mathcal{B}}(x)$; // project onto $\mathcal{B}$ only

11 **if** *β has not improved by ϵ for K iterations* **then break** ; // stop if converged

12 **return** $s \cdot \beta$; // certified lower/upper bound

For an ℓ_∞ ball, it clips z elementwise to the interval $[x_0 - \varepsilon\mathbf{1},\ x_0 + \varepsilon\mathbf{1}]$:

$$\Pi_{\mathcal{B}}(z) = \min\big(x_0 + \varepsilon\mathbf{1},\ \max(x_0 - \varepsilon\mathbf{1},\ z)\big) \tag{18}$$

where $\min$ and $\max$ here are elementwise operations.

CLAD adapts η with the Barzilai–Borwein step (Barzilai & Borwein, 1988), which takes larger steps in flat regions and smaller steps in sharply curved ones, avoiding manual tuning. With $dx_t = x_t - x_{t-1}$ and $y_t = q_t - q_{t-1}$, the Barzilai and Borwein update is

$$\eta_t = \max\left(0, \min\left(\frac{|dx_t^\top y_t|}{\|y_t\|_2^2}, 1\right)\right) \tag{19}$$

The Lagrangian at an unconverged iterate is not a sound bound, since $\mathcal{L}_s(x, \mu) \geq \mathrm{d}(\mu)$. CLAD instead evaluates, after every step, a *certified bound* from the tangent of $\mathcal{L}_s$ at x, minimized over $\mathcal{B}$ as in Eq. 16:

$$\hat{\mathrm{d}}(x, \mu) = \mathcal{L}_s(x, \mu) + \min_{x' \in \mathcal{B}} g^\top (x' - x) = \mathcal{L}_s(x, \mu) + g^\top (x_0 - x) - \varepsilon \|g\|_q \tag{20}$$

where $g = s \cdot A + \sum_j \mu_j \nabla h_j(x)$ is a subgradient of $\mathcal{L}_s(\cdot, \mu)$. Each h_j is convex, so $\mathcal{L}_s(\cdot, \mu)$ lies above its tangent and $\hat{\mathrm{d}}(x, \mu) \leq \mathrm{d}(\mu)$, which is sound by Thm. C.1 for every iterate x and every $\mu \geq 0$. Alg. 2 keeps the largest certified bound β, stops once β has not improved by more than ϵ for K consecutive iterations, and returns $s \cdot \beta$ (line 12). Stopping at any iteration is therefore safe.

3.3 SOUNDNESS, TIGHTNESS, AND CONVERGENCE OF CLAD

Every certified bound $\hat{\mathrm{d}}(x, \mu)$ is sound, so Alg. 2 can stop at any iteration and return a valid bound. Under the Slater condition, which requires a strictly feasible interior point, maximizing the dual function over $\mu \geq 0$ recovers the exact constrained optimum and is therefore *tight*. Apdx. C gives formal proofs of Thm. C.1, Thm. C.4, and resulting corollaries for halfspace and ℓ_2 ball constraints. For a simultaneous variant with a constant shared step and bounded multipliers, the averaged iterates converge to the constrained optimum at rate $O(1/\sqrt{T})$ (Thm. C.10); this concerns tightness only, as soundness holds at every iteration.

4 EVALUATION

Verification Benchmark We use four convolutional networks from Chiu et al. (2025): ConvSmall (M) trained on MNIST, and ConvSmall (C), ConvDeep, and ConvLarge trained on CIFAR-10. We

Tab. 1: Verified instances and average runtime per instance per network and method.

Network	IBP		ZONOTOPE		CROWN / DEEPPOLY		α-CROWN		SDP-CROWN		GCP-CROWN		CLAD	
	Verified	Time	Verified	Time	Verified	Time	Verified	Time	Verified	Time	Verified	Time	Verified	Time
ConvSmall (M)	0	0.00	216	0.00	264	0.04	297	4.94	297	14.24	300	18.81	**358**	19.23
ConvSmall (C)	0	0.00	222	0.00	270	0.04	312	5.24	312	15.59	315	28.77	**375**	22.55
ConvDeep	0	0.00	198	0.00	243	0.06	300	8.35	303	22.64	306	35.95	**354**	38.77
ConvLarge	0	0.00	0	0.10	0	0.20	18	30.78	18	63.70	18	72.37	**59**	78.05

do not use VNN-COMP instances, which are generally easy [2] for this comparison: most are already solved by simple abstractions such as IBP or ZONOTOPE, so they cannot differentiate among more precise methods. In contrast, these networks span 4 to 7 layers, 8 to 64 convolutional channels, and 55K to 2.47M parameters, and they are known to separate verifiers sharply (Wang et al., 2021; Chiu et al., 2025), *e.g.*, on the CIFAR-10 ConvLarge, α-CROWN, GCP-CROWN, and SDP-CROWN verify 2.5%, 6%, and 63.5% of the images, respectively (Chiu et al., 2025).

We generate more expressive instances using motion-blur perturbations from VeriDou (Duong et al., 2026b), whose convolutional parameterization covers a continuous range of blur angles, at six strengths for each network. We use structured perturbations noise because real corruptions such as blur change correlated pixels (Duong et al., 2026a;b). For every network and perturbation strength, we compare three setups: (i) ℓ_∞ interval (`linf`), (ii) that same interval intersects with a halfspace (`linf+hs`), (iii) interval intersects with an ℓ_2 ball (`linf+l2`). This yields 1,944 instances across the four networks and three property types. §D.2 gives more details on the benchmarks.

Comparison Baselines We compare CLAD with abstraction baselines including: IBP (Wang et al., 2018), ZONOTOPE, CROWN (Xu et al., 2020a)/DEEPPOLY (Singh et al., 2019), α-CROWN (Xu et al., 2020b), SDP-CROWN (Chiu et al., 2025), and GCP-CROWN (Zhang et al., 2022b). For a fair comparison, we give every baseline the minimum interval that contain the constrained input region of each property. CLAD receives the same tightened interval in addition to the constraint itself.

Comparison Metrics The *number of verified instances* is our primary measure of an abstraction's effectiveness, since more verified instances means robustness holds over a broader range of instances. An instance is verified if its output lower bound $l_{\text{out}} > 0$. For classification models, $l_{\text{out}} > 0$ when the margin $Y_i - Y_j > 0$ between the true class i and a competing class j.

4.1 RQ1: PERFORMANCE WITH EXISTING ABSTRACTION

Tab. 1 presents the verified instance counts and average per-instance runtimes for each method across the four convolutional networks. CLAD achieves the highest number of verified instances on each network, totaling 1,146, which improves 22% compared to the strongest baseline GCP-CROWN. The performance gap is most pronounced on ConvLarge, where CLAD verifies 59 instances, whereas α-CROWN, SDP-CROWN, and GCP-CROWN each verify fewer than one-third as many instances. IBP does not verify any instances, which indicates that the benchmark instances are non-trivial.

In terms of runtime, IBP, ZONOTOPE and CROWN require at most 0.2 seconds per instance but verify substantially fewer instances. while α-CROWN, SDP-CROWN, and GCP-CROWN take up to 72s per instance on ConvLarge. CLAD's runtime is close to that of GCP-CROWN while verifying 41 to 60 more instances per network. Consequently, the primal-dual solver increases verification precision while maintaining a runtime similar to that of the baseline abstraction domains.

4.2 RQ2: PERFORMANCE ON CONSTRAINED AND UNCONSTRAINED PROPERTIES

The leftmost cactus plot in Fig. 2 presents the runtime for verified instances of the unconstrained ℓ_∞ property. α-CROWN, SDP-CROWN, GCP-CROWN, and CLAD verify 309, 310, 313, and 316 instances, respectively. GCP-CROWN verifies most instances in approximately 0.1 seconds, but requires around 100 seconds for 50 challenging instances when the general cutting mechanism is activated. CLAD processes each instance in approximately 4 to 40 seconds. CLAD therefore achieves precision comparable to that of α-CROWN variants, with a similar per-instance compute time.

[2] Out of 2700 instances in regular track of VNN-COMP'24 (Brix et al., 2024), nearly 90% problems ($\sim$2400/2700) were solved with DEEPPOLY or α-CROWN.

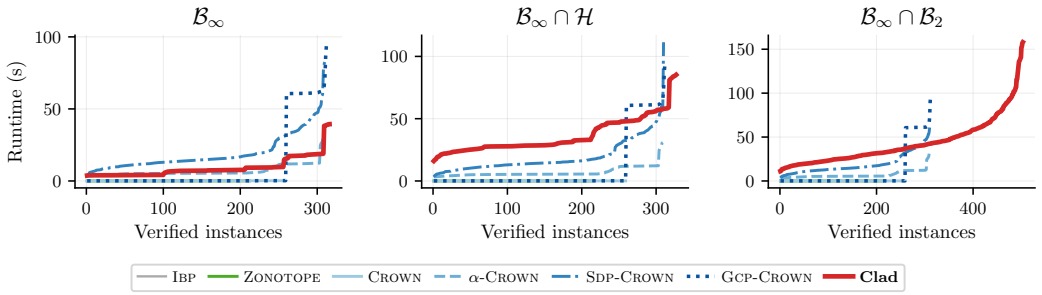


Fig. 2: Cactus plots of runtime per verified instances for each property type

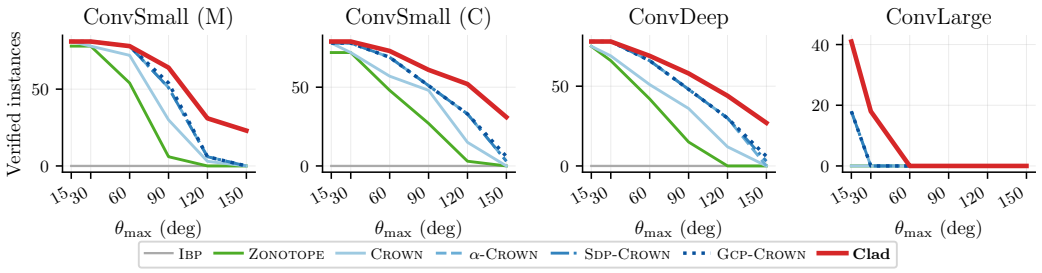


Fig. 3: Verified instances per motion-blur angle $\theta_{\max}$ on the four convolutional networks.

The advantage of CLAD becomes pronounced when the input region includes an additional constraint as demonstrated by the second and third cactus plots in Fig. 2. Each baseline curve remains consistent across all three panels so even with a tightened interval no baseline verifies more instances than under the unconstrained property. On $\mathcal{B}_\infty \cap \mathcal{H}$, CLAD verifies approximately 330 instances, compared to approximately 313 for the strongest baselines. On $\mathcal{B}_\infty \cap \mathcal{B}_2$, CLAD verifies approximately 500 instances, representing a sharp increase of roughly 60% over GCP-CROWN. With additional contrains, CLAD requires additional time for its Lagrangian concretization step, resulting in its cactus curve lying slightly above the baselines. The ℓ_2 ball reduces the feasible region most significantly and therefore demonstrates the clearest benefit - CLAD requires at most approximately 60% time per instance while also verifies 60% more instances than GCP-CROWN.

4.3 RQ3: PERFORMANCE ON DIFFERENT PERTURBATION RADII

Fig. 3 presents the number of verified instances for each method as the maximum motion-blur angle $\theta_{\max}$ increases. At 15° and 30° on ConvSmall and ConvDeep, α-CROWN variants is the same as CLAD because the small input space remains manageable for all these abstractions. As $\theta_{\max}$ increases and the input space expands, the performance gap between CLAD and the baseline methods becomes more pronounced. At 150° on ConvSmall (C) and ConvDeep, GCP-CROWN verifies only six instances each, whereas CLAD verifies 31 and 27 instances, respectively. ConvLarge, which contains 2.47M parameters, presents greater difficulty. α-CROWN, SDP-CROWN, and GCP-CROWN each verify 18 instances at 15° and none at larger angles, whereas CLAD verifies 41 instances at 15° and 18 at 30°. Therefore, the advantage of CLAD observed in §4.2 increases as the perturbation strength and model size grows.

5 CONCLUSION

We presented CLAD, an abstract domain that retains additional input constraints in a Lagrangian and uses projected primal dual concretization to tighten neuron bounds. CLAD handles convex constraints and their intersections, returns a sound bound at every iteration, and is tight under the Slater condition. Future work includes supporting other activations, integration with branch and bound (Bunel et al., 2020b), and extending to nonconvex perturbations (Duong et al., 2026a).

AI USE STATEMENT

In this work, we used generative AI tools (Claude) to assist in writing the Lean proofs that mechanize Thm. C.1 (soundness of the dual bound for every $\mu \geq 0$), Thm. C.4 (tightness under the Slater condition), Thm. C.6 (tightness for a halfspace constraint), and Thm. C.8 (tightness for an ℓ_2-ball constraint).

We have not used generative AI tools for help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, design or provide feedback on research methodology or experiments, propose or refine hypotheses, implement methods, interpret results, and the rest of the required disclosure tasks are not applicable to this work.

Additionally, we used generative AI tools (Cursor) to generate code to plot result table and figures. We have reviewed all AI-assisted work. For example, we read the python code LLM created for the figures and checked against the value in raw data files. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

All authors have read and adhere to the ICLR 2027 Code of Ethics.

REPRODUCIBILITY STATEMENT

CLAD implementation is available at `https://anonymous.4open.science/r/clad`.

REFERENCES

Kenneth Joseph Arrow, Leonid Hurwicz, Hirofumi Uzawa, Hollis Burnley Chenery, Selmer Johnson, and Samuel Karlin. *Studies in linear and non-linear programming*. Stanford University Press Stanford, 1958.

Junjie Bai, Fang Lu, and Ke Zhang. ONNX Open neural network exchange, 2025. URL `https://onnx.ai`.

Stanley Bak. nnenum: Verification of ReLU Neural Networks with Optimized Abstraction Refinement. In *NASA Formal Methods Symposium*, pp. 19–36. Springer, 2021. doi: 10.1007/978-3-030-76384-8_2.

Jonathan Barzilai and Jonathan M. Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988. doi: 10.1093/imanum/8.1.141.

Žiga Bizjak, June Ho Choi, Wonhyoung Park, and Žiga Špiclin. Deep learning based modality-independent intracranial aneurysm detection. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 760–769. Springer, 2022. doi: 10.1007/978-3-031-16437-8_73.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T. Johnson. The Fourth International Verification of Neural Networks Competition (VNN-COMP 2023): Summary and Results, 2023.

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. doi: 10.48550/arXiv.2412.19985.

Rudy Bunel, Alessandro De Palma, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip Torr, and M Pawan Kumar. Lagrangian decomposition for neural network verification. In *Conference on Uncertainty in Artificial Intelligence*, pp. 370–379. PMLR, 2020a.

Rudy Bunel, P Mudigonda, Ilker Turkaslan, P Torr, Jingyue Lu, and Pushmeet Kohli. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(2020), 2020b. URL `https://dl.acm.org/doi/10.5555/3455716.3455758`.

Mohamad Fares El Hajj Chehade, Wenting Li, Brian Wesley Bell, Russell Bent, Saif R. Kazi, and Hao Zhu. LEVIS: Large exact verifiable input spaces for neural networks. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 7634–7647. PMLR, 13–19 Jul 2025.

Hong-Ming Chiu, Hao Chen, Huan Zhang, and Richard Y. Zhang. SDP-CROWN: Efficient bound propagation for neural network verification with tightness of semidefinite programming. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 10449–10468. PMLR, 13–19 Jul 2025.

Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pp. 238–252, 1977. doi: 10.1145/512950.512973.

Alessandro De Palma, Rudy Bunel, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip H. S. Torr, and M. Pawan Kumar. Improved branch and bound for neural network verification via lagrangian decomposition. *arXiv preprint arXiv:2104.06718*, 2021.

Hai Duong and ThanhVu Nguyen. Neuralsat: Scaling constraint solving for dnn verification (competition contribution). In *International Symposium on AI Verification*, pp. 253–259. Springer, 2025. doi: 10.1007/978-3-031-99991-8_14.

Hai Duong, Dong Xu, Thanhvu Nguyen, and Matthew B. Dwyer. Harnessing neuron stability to improve dnn verification. *Proc. ACM Softw. Eng.*, 1(FSE), 2024. doi: 10.1145/3643765.

Hai Duong, ThanhVu Nguyen, and Matthew B Dwyer. Generating and checking dnn verification proofs. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a.

Hai Duong, ThanhVu Nguyen, and Matthew B Dwyer. Neuralsat: A high-performance verification tool for deep neural networks. In *International Conference on Computer Aided Verification*, pp. 409–423. Springer, 2025b. doi: 10.1007/978-3-031-98679-6_19.

Hai Duong, David Shriver, ThanhVu Nguyen, and Matthew B Dwyer. Compositional neural network verification via assume-guarantee reasoning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025c.

Hai Duong, Thanh Le, Lam Nguyen, and ThanhVu Nguyen. Verifying Structural Robustness of Deep Neural Network. *Proceedings of the ACM on Software Engineering*, 3(FSE), 2026a. doi: 10.1145/3797095.

Hai Duong, Lam Nguyen, Thanh Le, and ThanhVu Nguyen. Verifying neural network robustness with dual perturbations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 27916–27925, 2026b.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *UAI*, volume 1, pp. 3, 2018.

Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1625–1634, 2018.

Claudio Ferrari, Mark Niklas Mueller, Nikola Jovanović, and Martin Vechev. Complete Verification via Multi-Neuron Relaxation Guided Branch-and-Bound. In *International Conference on Learning Representations*, 2022. doi: 10.48550/arXiv.2205.00263.

Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 37:100270, 2020. doi: 10.1016/j.cosrev.2020.100270.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: a calculus for reasoning about deep neural networks. *Formal Methods in System Design*, 60(1):87–116, 2022. doi: 10.1007/s10703-021-00363-7.

Konstantin Kaulen, Tobias Ladner, Stanley Bak, Christopher Brix, Hai Duong, Thomas Flinkow, Taylor T. Johnson, Lukas Koller, Edoardo Manino, ThanhVu H Nguyen, and Haoze Wu. The 6th international verification of neural networks competition (vnn-comp 2025): Summary and results, 2025. URL https://arxiv.org/abs/2512.19007.

Suhas Kotha, Christopher Brix, J. Zico Kolter, Krishnamurthy Dvijotham, and Huan Zhang. Provably bounding neural network preimages. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Angelia Nedić and Asuman Ozdaglar. Subgradient methods for saddle-point problems. *Journal of Optimization Theory and Applications*, 142(1):205–228, 2009. doi: 10.1007/s10957-009-9522-7.

Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009. doi: 10.1137/070704277.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

PyRAT. A tool to analyze the robustness and safety of neural networks, 2024. URL https://pyrat-analyzer.com/.

Hao Shao, Letian Wang, Ruobing Chen, Hongsheng Li, and Yu Liu. Safety-enhanced autonomous driving using interpretable sensor fusion transformer. In *Conference on Robot Learning*, pp. 726–737. PMLR, 2023.

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. *Advances in Neural Information Processing Systems*, 31, 2018.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, 2019. doi: 10.1145/3291645.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1599–1614, 2018. URL https://dl.acm.org/doi/10.5555/3277203.3277323.

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. In *Advances in Neural Information Processing Systems*, volume 34, pp. 29909–29921, 2021.

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*, pp. 249–264. Springer, 2024. doi: 10.1007/978-3-031-65630-9_13.

Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33:1129–1141, 2020a.

Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. *arXiv preprint arXiv:2011.13824*, 2020b. doi: 10.48550/arXiv.2011.13824.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems*, volume 31, pp. 4939–4948, Montréal, Canada, 2018. Curran Associates, Inc.

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. In *Advances in Neural Information Processing Systems*, volume 35, 2022a.

Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. General cutting planes for bound-propagation-based neural network verification. *Proceedings of the 36th International Conference on Neural Information Processing Systems*, 2022b. URL https://dl.acm.org/doi/10.5555/3600270.3600391.

Xiyue Zhang, Benjie Wang, and Marta Kwiatkowska. Provable preimage under-approximation for neural networks. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 3–23. Springer, 2024. doi: 10.1007/978-3-031-57256-2_1.

Duo Zhou, Christopher Brix, Grani A Hanasusanto, and Huan Zhang. Scalable neural network verification with branch-and-bound inferred cutting planes. *arXiv preprint arXiv:2501.00200*, 2024.

Duo Zhou, Jorge Chavez, Hesun Chen, Grani A. Hanasusanto, and Huan Zhang. Clip-and-verify: Linear constraint-driven domain clipping for accelerating neural network verification. In *Advances in Neural Information Processing Systems*, 2025.

A Background

A.1 Neural Network Verification (NNV)

Definition A.1 (NNV Problem)**.** Given a DNN N and a property (or specification) ϕ, the NNV problem asks whether ϕ is a valid property of N. Typically, ϕ takes the form $\phi_{in} \Rightarrow \phi_{out}$, where ϕ_{in} is a property over the inputs of N and ϕ_{out} is a property over the outputs of N.

Modern DNN techniques (Wang et al., 2021; Duong et al., 2024; 2025b; Zhang et al., 2022b; Zhou et al., 2024; Bak, 2021; Wu et al., 2024; Ferrari et al., 2022) treat this verification problem as a satisfiability problem by encoding the DNN N and the property ϕ as a logical formula:

$$N \wedge \phi_{in} \wedge \neg\phi_{out} \tag{21}$$

If Eq. 21 is unsatisfiable (UNSAT), the considered property holds. Otherwise, it is satisfiable (SAT) and a counterexample exists that disproves the property.

A.2 Linear Relaxation

Linear relaxation bounds each intermediate neuron v_i by affine functions of the network inputs x:

$$\underline{A}_i x + \underline{b}_i \;\le\; v_i \;\le\; \overline{A}_i x + \overline{b}_i \tag{22}$$

For linear layers, bounds propagate exactly. For an unstable ReLU with pre-activation $\hat{z} \in [l, u]$, $l < 0 < u$, and output $z = \mathrm{ReLU}(\hat{z})$, Crown (Xu et al., 2020a) and DeepPoly (Singh et al., 2019) over-approximate the activation by:

$$\alpha\,\hat{z} \;\le\; z \;\le\; \bar{d}\,\hat{z} + \bar{b} \qquad \bar{d} = \tfrac{u}{u-l} \qquad \bar{b} = -\tfrac{lu}{u-l} \qquad \alpha \in [0,1] \tag{23}$$

Stable neurons ($l \ge 0$ or $u \le 0$) need no relaxation. These per-neuron relaxations are back-substituted through preceding linear layers until Eq. 22 is expressed entirely in terms of x, and then *concretized* by optimizing over input space $\mathcal{B}$:

$$l_i = \min_{x \in \mathcal{B}} (\underline{A}_i x + \underline{b}_i), \qquad u_i = \max_{x \in \mathcal{B}} (\overline{A}_i x + \overline{b}_i) \tag{24}$$

For a standard ℓ_∞ ball, these reduce to closed-form expressions via Hölder's inequality Wang et al. (2021).

B Related Work

Existing abstractions form a spectrum of speed-accuracy tradeoffs, *e.g.*, better speed by IBP (Wang et al., 2018), ZONOTOPE (Singh et al., 2018), CROWN (Zhang et al., 2018) and DEEPPOLY (Singh et al., 2019), or better precision by α-CROWN (Xu et al., 2020a), GCP-CROWN (Zhang et al., 2022b). All of these methods, however, compute bounds by optimizing over the enclosing ℓ_∞ interval and discard any additional input constraints such as halfspace or ℓ_2-ball intersections. CLAD addresses this gap by incorporating such constraints directly into each bound propagation step.

Lagrangian duality itself has a long history in NNV. Dvijotham et al. (2018) dualize the layer equations of the entire network and optimize the multipliers by subgradient ascent, Bunel et al. (2020a) instead dualize the standard LP relaxation which De Palma et al. (2021) then integrate into a complete branch-and-bound verifier. Because these dualize the network itself the dual dimension grows with the network size which isn't as scalable as linear relaxation abstraction domains since linear relaxation abstractions keep closed-form bound propagation and dualize only a small set of side constraints, *e.g.*, β-CROWN for branch-and-bound splits (Wang et al., 2021), GCP-CROWN for cutting planes (Zhang et al., 2022a), and INVPROP for output constraints (Kotha et al., 2023). Closest to CLAD, Clip-and-Verify (Zhou et al., 2025) derives linear constraints in the input space from hidden-neuron branch-and-bound splits and output specifications, and uses them to clip the input box and retighten intermediate bounds. For a single linear constraint over a box, it solves the same one-dimensional piecewise-linear dual that CLAD uses for a halfspace. CLAD differs in the constraints it handles: they are part of the input specification rather than derived during verification, and they may be any convex function, such as an ℓ_2 ball, whereas Clip-and-Verify handles only linear inequalities. CLAD dualizes only the M input constraints during concretization (Alg. 2), thus, yields a dual of dimension M independent of the network size, which also remains scalable.

C Analysis of CLAD

C.1 Soundness and Tightness

The results in this appendix are stated for the upper bound u_i in the maximization form, which matches the Lean mechanization. The lower-bound form of §3.2 follows by replacing $Ax + b$ with $-(Ax + b)$.

An abstraction is *sound* if the bounds it returns always over-approximate and never underestimate the true range of a neuron. This means the value the abstraction returned for the upper bound must never fall below the true constrained maximum and the lower bound never reach above the true constrained minimum. For CLAD, soundness holds for *every* Lagrange multiplier value μ, and every certified bound $\hat{\mathrm{d}}(x, \mu)$ of Eq. 20 is bounded by $\mathrm{d}(\mu)$, which means Alg. 2 may stop at any iteration and still return a valid bound.

Theorem C.1 (CLAD soundness)**.** *For every $\mu \geq 0$,*

$$\mathrm{d}(\mu) \;=\; \max_{x \in \mathcal{B}} \mathcal{L}(x, \mu) \;\geq\; \max_{x \in \mathcal{B} \cap \mathcal{H}} Ax + b \;=\; u_i \tag{25}$$

Proof. At every feasible point $x \in \mathcal{B} \cap \mathcal{H}$, each constraint function satisfies $h_j(x) \leq 0$. Thus, every penalty term $-\mu_j h_j(x)$ is non-negative and the Lagrangian dominates the objective: $\mathcal{L}(x, \mu) = Ax + b - \mu_j h_j(x) \geq Ax + b$. Maximizing over the larger set $\mathcal{B} \supseteq \mathcal{B} \cap \mathcal{H}$ can only increase this value, giving $\mathrm{d}(\mu) \geq u_i$. □

Remark C.2 (Lean mechanization). Thm. C.1 is fully mechanized in Lean 4 against Mathlib as `CLAD.primalVal_le_dualFn` in `lean/CLAD/Soundness.lean` of our released code. Lean mechanizes the weak-duality step $\mathrm{d}(\mu) \geq u_i$ for the exact dual function. The tangent step relating the certified bound of Eq. 20 to $\mathrm{d}(\mu)$ is the one-line convexity argument of §3.2; it is not mechanized, and neither is floating-point arithmetic. Every mechanized result in this subsection is `sorry`-free and depends only on Lean's three standard axioms, *e.g.*, `propext`, `Classical.choice` and `Quot.sound`. The mechanization is more general than the statement above: the first proof step, `CLAD.le_lagrangian_of_mem_feasible`, establishes $Ax + b \leq \mathcal{L}(x, \mu)$ at every feasible x using only $\mu \geq 0$ and $h_j(x) \leq 0$, with no convexity, no continuity, and no linearity of the objective. The mechanized statement carries two side conditions, $\mathcal{B} \cap \mathcal{H} \neq \emptyset$ and $\mathcal{L}(\cdot, \mu)$ bounded above on $\mathcal{B}$;

these are artifacts of encoding $\max$ as a real-valued supremum, not mathematical content, and both hold automatically for the compact $\mathcal{B}$ of our setting.

The condition required for CLAD to achieve tightness follows the classical *Slater condition* (Boyd & Vandenberghe, 2004, §5.2).

Definition C.3 (Slater condition)**.** Assume the M convex constraint functions can be ordered such that $h_1, \ldots, h_k$ are non-affine and $h_{k+1}, \ldots, h_M$ are affine. The Slater condition holds for the constrained problem $\max_{x \in \mathcal{B} \cap \mathcal{H}} Ax + b$ if there exists a point $\hat{x}$ in the interior of the input domain $\operatorname{int} \mathcal{B}$ such that $h_j(\hat{x}) < 0$ for every $j \in \{1, \ldots, k\}$ and $h_j(\hat{x}) \leq 0$ for every $j \in \{k+1, \ldots, M\}$.

Theorem C.4 (Tightness of CLAD for convex constraint intersections)**.** *Let the radius $\varepsilon > 0$, let each constraint function h_j be convex and continuous on $\mathbb{R}^D$, and suppose there exists a point in the relative interior of the base interval $\hat{x} \in \operatorname{int} \mathcal{B}$ satisfying $h_j(\hat{x}) < 0$ for every non-affine constraint function $h_j(.), \forall j \in \{1, \ldots, k\}$ and $h_j(\hat{x}) \leq 0$ for every affine constraint function $h_j, \forall j \in \{k+1, \ldots, M\}$. Then the dual bound $\mathrm{d}(\mu)$ attains the minimal upper bound $u_i^\star$ at its minimum over the multiplier $\mu \geq 0$:*

$$\min_{\mu \geq 0} \mathrm{d}(\mu) \;=\; \max_{x \in \mathcal{B} \cap \mathcal{H}} Ax + b \;=\; u_i^\star \tag{26}$$

and this minimum is attained at some optimal multiplier $\mu^\star \geq 0$.

Proof. The assumptions fit Slater's condition in Thm. C.3. Thus, strong duality holds, the duality gap is zero, and there exists some multiplier $\mu^\star \geq 0$ attaining this optimum. □

Remark C.5 (Lean mechanization). Thm. C.4 is mechanized as `CLAD.dualFn_eq_primalVal_of_slater` in `lean/CLAD/Tightness.lean`. The mechanized proof routes through Sion's minimax theorem (`Sion.exists_isSaddlePointOn` in Mathlib) applied to $\mathcal{L}$ on $[0, \kappa]^M \times \mathcal{B}$, where the Slater margin supplies an a priori bound κ on any multiplier whose dual value is at most $\mathrm{d}(0)$. The conclusion is the full claim of equation 26: both the zero duality gap and the attainment of the minimum over *all* $\mu \geq 0$, not merely over the truncation used inside the proof. Two differences from the statement above are worth recording. First, the objective is assumed only concave and continuous on $\mathcal{B}$; the affine $Ax + b$ is a special case, so the mechanized result is a generalization, and the unused hypothesis $\varepsilon > 0$ is dropped. Second, the mechanized Slater hypothesis asks for a uniform strict margin $h_j(\hat{x}) \leq -\delta < 0$ at *every* constraint, *e.g.*, the case $k = M$ of Thm. C.3; over finitely many constraints this is equivalent to requiring $h_j(\hat{x}) < 0$ for each j. The affine relaxation of Thm. C.3 is therefore not used in the general theorem, but it is recovered exactly where the paper applies it, in Thm. C.6.

Thm. C.4 applies directly to the two constraint families of §3.1:

Corollary C.6 (Halfspace tightness)**.** *Let $\mathcal{H} = \{x : cx \leq d\}$ with $c \neq 0$. If $\operatorname{int} \mathcal{B} \cap \mathcal{H} \neq \emptyset$, then $\min_{\mu \geq 0} \mathrm{d}(\mu) = u_i^\star$.*

Proof. The constraint function $h(x) = cx - d$ is affine, so feasibility of $\hat{x}$ satisfies the Slater condition. □

Remark C.7 (Lean mechanization). Thm. C.6 is mechanized as `CLAD.tight_halfspace` in `lean/CLAD/Tightness.lean`, under exactly the non-strict hypothesis $c\hat{x} \leq d$ stated above. The strict margin demanded by `CLAD.dualFn_eq_primalVal_of_slater` is manufactured rather than assumed: `CLAD.exists_strict_slater_of_mem_interior` perturbs $\hat{x}$ to $\hat{x} - tv$ for a direction v with $cv > 0$ and small $t > 0$, which remains in $\mathcal{B}$ because $\hat{x} \in \operatorname{int} \mathcal{B}$ and strictly decreases $cx - d$ because $c \neq 0$. This is the step that recovers the affine relaxation of Thm. C.3 discussed in Thm. C.5.

Corollary C.8 (ℓ_2-ball tightness)**.** *Let $\mathcal{H} = \{x : \|x - x_c\|_2 \leq r\}$ with $r > 0$. If there exists $\hat{x} \in \operatorname{int} \mathcal{B}$ with $\|\hat{x} - x_c\|_2 < r$, then $\min_{\mu \geq 0} \mathrm{d}(\mu) = u_i^\star$.*

Proof. The point $\hat{x}$ is a strictly feasible Slater point for $h(x) = \|x - x_c\|_2 - r$. □

Remark C.9 (Lean mechanization). Thm. C.8 is mechanized as `CLAD.tight_l2Ball` in `lean/CLAD/Tightness.lean`, instantiating the Slater margin as $\delta = r - \|\hat{x} - x_c\|_2$, which is positive precisely by the hypothesis $\|\hat{x} - x_c\|_2 < r$. Convexity of the constraint is discharged from Mathlib's `convexOn_dist`.

C.2 CONVERGENCE

Theorem C.10 (Convergence of the primal-dual concretization)**.** *Let each constraint function h_j be convex and continuous, and suppose the Slater condition of Thm. C.4 holds, so that the interval $[0,\kappa]^M$ contains the optimal multiplier $\mu^\star$. With the constant step size $\alpha = R_{\mathcal{Z}}/(G\sqrt{T})$ shared by the primal and dual updates of the simultaneous variant of Alg. 2, the iterate averages $\bar{x} = \frac{1}{T}\sum_{t\le T} x_t$ and $\bar{\mu} = \frac{1}{T}\sum_{t\le T}\mu_t$ satisfy*

$$\mathrm{d}(\bar{\mu}) - u_i^\star \;\le\; \frac{R_{\mathcal{Z}}\, G}{\sqrt{T}},$$

where $R_{\mathcal{Z}} = \sqrt{R^2+\kappa^2 M}$ is the diameter of $\mathcal{Z} = \mathcal{B}\times[0,\kappa]^M$; $G_x = \sup_{x,\mu}\|A - \sum_j \mu_j \nabla h_j(x)\|_2$, $G_\mu = \sup_{x\in\mathcal{B}}\|(h_1(x),\dots,h_M(x))\|_2$, and $G = \sqrt{G_x^2 + G_\mu^2}$.

Proof. The bound follows by adding two regret guarantees: the primal step matches the standard rate for supergradient ascent on the concave function $\mathcal{L}(\cdot,\mu_t)$, and the dual step matches the analogous rate for subgradient descent on $\mathcal{L}(x_t,\cdot)$ (Nedić & Ozdaglar, 2009; Nemirovski et al., 2009); the two regrets add because their common term $\mathcal{L}(x_t,\mu_t)$ cancels. Averaging via Jensen's inequality and identifying the saddle value with $u_i^\star$ using Thm. C.1 and Thm. C.4 yields the stated rate. □

This guarantee concerns an idealized simultaneous variant. Alg. 2 uses sequential updates, the Barzilai–Borwein step, unbounded multipliers, and early stopping; its soundness rests on the certified bound of Eq. 20, not on convergence.

D IMPLEMENTATION DETAILS

CLAD is implemented in Python and uses the PyTorch framework (Paszke et al., 2019). It accepts neural networks in ONNX format (Bai et al., 2025) and verification properties in VNNLib format.The ONNX network is parsed into a directed acyclic graph, after which the bound-propagation pass from Alg. 1 is executed. Each constraint is implemented in hinge form $\phi_j = \max(0, h_j)$. Its subgradient is computed in closed form for the interval and halfspace constraints and with PyTorch's autograd for the ℓ_2 ball.

CLAD supports general convex constraints, and we consider three representative constraint types in our benchmark (§D.2): halfspace ($c_j x \le d_j$, constraint function $c_j x - d_j$), ℓ_2 ball ($\|x - x_c\|_2 \le r$, constraint function $\|x - x_c\|_2 - r$), and ℓ_∞ interval (represented directly as variable bounds during the primal projection step). The primal-dual solver for constraint-aware concretization (Alg. 2) is executed for a maximum of T=500 iterations, with a primal step size of η=0.01 and a dual step size of τ=0.1. Multipliers start at $\mu_0 = \|A\|_2$, the norm of the bound's coefficient vector. Besides $\hat{\mathrm{d}}(x,\mu)$ at every iterate, CLAD folds into β the μ=0 bound and, for each constraint, the exact maximizer of the certified bound over that constraint's multiplier alone; all are certified bounds, so their maximum is returned. Early termination occurs when β has not improved by more than $\epsilon = 3\times10^{-4}$ for K=10 consecutive iterations. All computations use floating point without directed rounding, as in the baselines. GCP-CROWN runs α-CROWN followed by CPLEX cut generation with 100 cuts and a 120-second budget per instance.

D.1 RUNTIME AND ENVIRONMENT

Our experiments were conducted on a Linux system equipped with an AMD 12-core CPU, 128 GB of RAM, and an NVIDIA GeForce RTX 4080 GPU (16 GB VRAM), which is leveraged by all methods (CLAD and baselines). Runtime is the average wall-clock time, in seconds, per instance. Because every method computes bounds in a single propagation pass (§4), all instances run to completion and no timeouts occur. Runtime therefore reflects the full computational cost of each abstraction on an instance.

D.2 VERIFICATION BENCHMARK

Our evaluation uses the standard ℓ_∞-interval verification setting and network architectures commonly used in NNV evaluation (*e.g.*, VNN-COMPs (Kaulen et al., 2025; Brix et al., 2024; 2023)). We therefore generate instances, with the *same* networks, at harder perturbation radii where the choice of abstract domain is decisive.

The benchmark uses four convolutional network architectures from existing work (Chiu et al., 2025): a ConvSmall trained on MNIST, and a ConvSmall, ConvDeep, and ConvLarge trained on CIFAR-10. We use motion-blur structured perturbations from VeriDou (Duong et al., 2026b) instead of per-pixel ℓ_∞ perturbations, for three reasons. First, standard local robustness perturbs each pixel independently inside an ℓ_∞ box, but real image corruptions such as motion blur, camera shake, and filtering are *structured*: each output pixel is a weighted combination of its neighbors, so pixels move together (Duong et al., 2026a;b). Second, an ℓ_∞ box cannot express this coupling exactly. Enclosing a structured perturbation in a pixel box admits images the transformation never produces, which leads to spurious counterexamples (Duong et al., 2026a). Third, structured perturbations expose weaknesses that ℓ_∞ noise misses: networks that appear highly robust under independent pixel noise are falsified on up to 99% of instances once convolutional perturbations are added (Duong et al., 2026b).

VeriS (Duong et al., 2026a) and VeriDou (Duong et al., 2026b) make such perturbations verifiable by standard tools. They prepend an affine layer that maps the perturbation parameters to the perturbed image, which turns structural robustness into ordinary local robustness over those parameters. We follow VeriDou's universal convolutional parameterization. Each entry of a 5×5 blur kernel is an independent variable, bounded in $[0, 1/5]$ on the positions covered by blur lines in the angle range and fixed to 0 elsewhere. This covers a continuous range of angles and is strictly more expressive than restricted formulations that interpolate between a few fixed kernels. As a result, the `linf` property is a box over kernel entries rather than over pixels, and the halfspace and ℓ_2 constraints restrict which blur kernels are allowed. We sweep the kernel angle over $[0^\circ, \theta_{\text{max}}]$ for six strengths $\theta_{\text{max}} \in \{15^\circ, 30^\circ, 60^\circ, 90^\circ, 120^\circ, 150^\circ\}$.

For each (model, perturbation strength) pair, we evaluate three property types: the base ℓ_∞ box alone (`linf`), the box intersected with a halfspace constraint (`linf+hs`), and the box intersected with an ℓ_2 ball constraint (`linf+l2`). For each base perturbation, the halfspace is $\sum x_i \leq \sum c_i$ (a hyperplane through the box center) and the ℓ_2 ball is centered at the upper corner $c + r$ with radius $\|r\|_2$, so all three property types share the same perturbation region and are directly comparable. We sample 3 correctly-classified test images per combination, each yielding 9 output disjunctions (one per non-true class), giving $4 \times 6 \times 3 \times 3 \times 9 = 1{,}944$ instances in total.