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

Duo Zhou*, Jorge Chavez*, Hesun Chen, Grani A. Hanasusanto, Huan Zhang

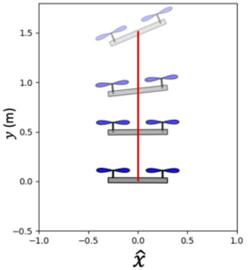
*co-first authors

University of Illinois Urbana-Champaign

Clip-and-Verify is part of α,β-CROWN, Winner of International Verification of Neural Networks Competitions (VNN-COMP 2021–2025)

The Problem: We Need Provably Safe AI

- Neural networks are used in safety-critical systems (autonomous driving, medical).
- Testing isn't enough, we need **formal proofs** of safety.
- **Challenge:** Formal verification is NP-complete and slow.
- **Goal:** Make the proof process dramatically faster.
- Existing verifiers compute a lower bound, $\underline{f}(\boldsymbol{x}) \leq f(\boldsymbol{x})$, to verify Eq. (1).



Yang et al. 2024

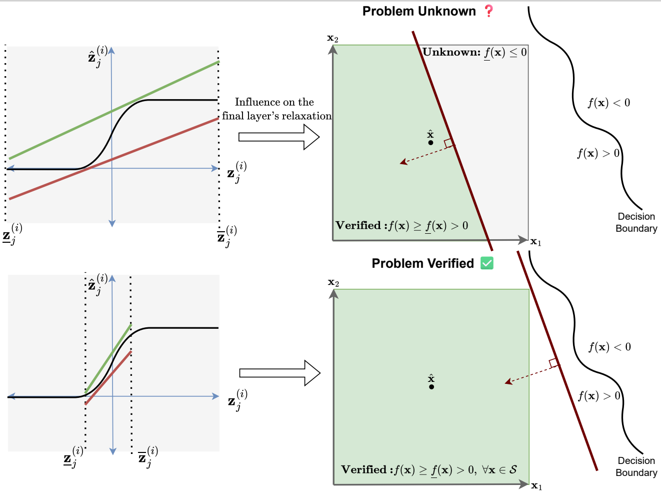
We must prove the quadrotor can stabilize itself from a set, $\mathcal{S}$, of starting points.

Canonical form of neural network verification

$$\text{Given } \boldsymbol{x} \in \mathcal{S}, \text{ is } f(\boldsymbol{x}) \leq 0 \text{ satisfiable?} \tag{1}$$

Contributions of **Clip-and-Verify**

- Despite their impact on the overall relaxation, $\underline{f}(\mathbf{x})$, SOTA verifiers struggle to efficiently tighten *intermediate-layer* bounds.
- Clip-and-Verify introduces two GPU-accelerated methods to refine input domains and tighten bounds at *any* layer.



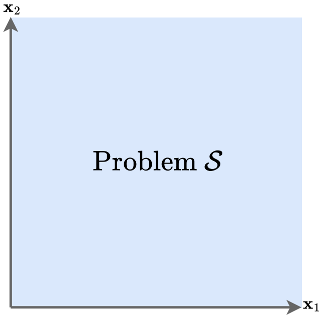
Lower bound, $\underline{\mathbf{z}}_j^{(i)} := \min_{\mathbf{x}\in\mathcal{S}} \underline{\mathbf{A}}_j^{(i),\top}\mathbf{x} + \underline{\mathbf{c}}_j^{(i)}$ Upper bound, $\overline{\mathbf{z}}_j^{(i)} := \max_{\mathbf{x}\in\mathcal{S}} \overline{\mathbf{A}}_j^{(i),\top}\mathbf{x} + \overline{\mathbf{c}}_j^{(i)}$

Lower bound relaxation: $\underline{\mathbf{a}}^{(L),\top}\mathbf{x} + \underline{\mathbf{c}}^{(L)} \le \mathbf{f}(\mathbf{x})$ Ground truth output: $\mathbf{f}(\mathbf{x})$

Preliminaries: How Verification Works

Branch-and-Bound (BaB): split hard regions; bound each subproblem; repeat.

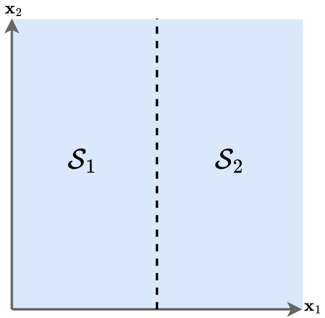
1. Start with one large box of inputs.
2. If Eq. (1) can't be verified (i.e., $\exists \boldsymbol{x} \in \mathcal{S},\ \underline{f}(\boldsymbol{x}) \leq 0$), **branch**: split the box.
3. **Bound**: compute bounds for each new box.
4. Recurse until the property is proved or refuted for *all* subproblems.



Preliminaries: How Verification Works

Branch-and-Bound (BaB): split hard regions; bound each subproblem; repeat.

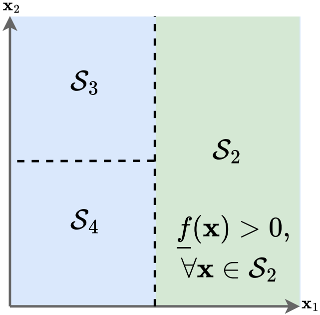
1. Start with one large box of inputs.
2. If Eq. (1) can't be verified (i.e., $\exists \boldsymbol{x} \in \mathcal{S},\ \underline{f}(\boldsymbol{x}) \leq 0$), **branch**: split the box.
3. **Bound**: compute bounds for each new box.
4. Recurse until the property is proved or refuted for *all* subproblems.



Preliminaries: How Verification Works

Branch-and-Bound (BaB): split hard regions; bound each subproblem; repeat.

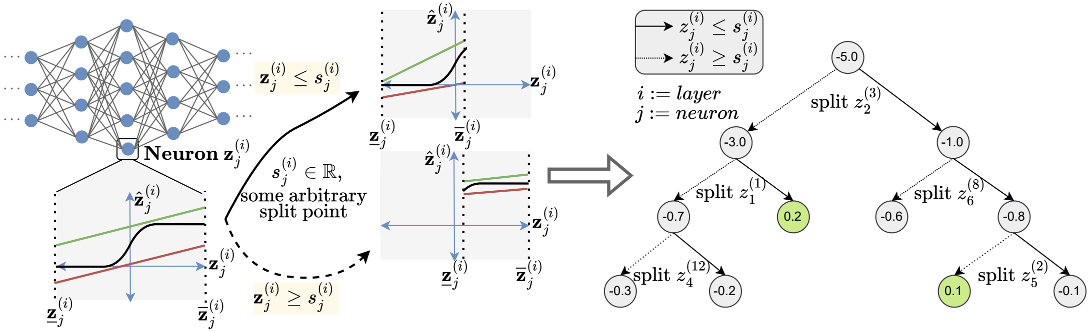
1. Start with one large box of inputs.
2. If Eq. (1) can't be verified (i.e., $\exists \mathbf{x} \in \mathcal{S},\ \underline{f}(\mathbf{x}) \leq 0$), **branch**: split the box.
3. **Bound**: compute bounds for each new box.
4. Recurse until the property is proved or refuted for *all* subproblems.



Preliminaries: How Verification Works

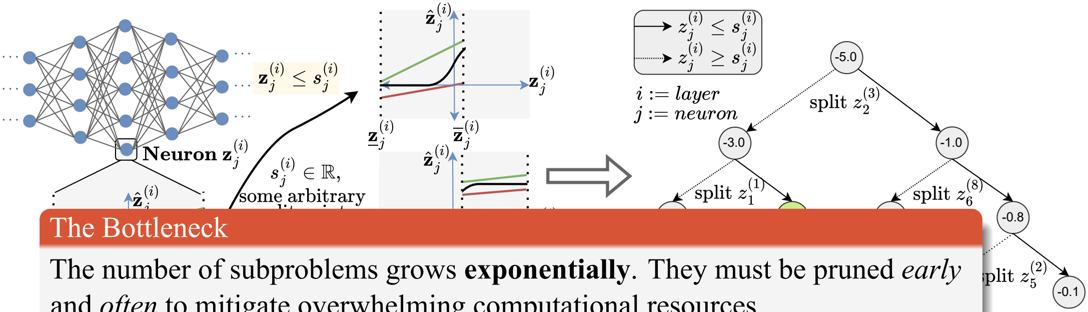
Branch-and-Bound (BaB): Beyond the input space.

- Pre-activation neurons may be split for general activations, (e.g. $s_j^{(i)} = 0$ for ReLU) to branch off subproblems.
- Clip-and-Verify generalizes to input splits or splits on any activation function.



Branch-and-Bound (BaB): Beyond the input space.

- Pre-activation neurons may be split for general activations, (e.g. $s_j^{(i)} = 0$ for ReLU) to branch off subproblems.
- Clip-and-Verify generalizes to input splits or splits on any activation function.

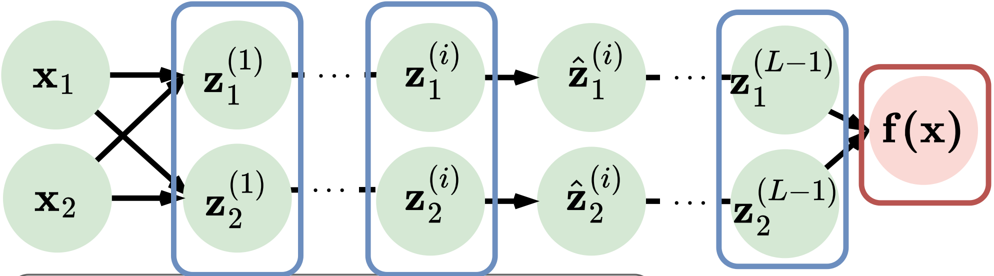


The Bottleneck

The number of subproblems grows **exponentially**. They must be pruned *early* and *often* to mitigate overwhelming computational resources.

Our Core Insight: Don't Waste Information

- Linear bound propagation provides linear relaxations at every layer.

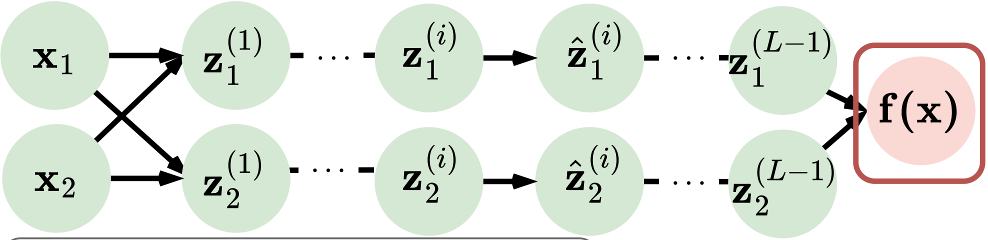


$$\mathbf{f(x)} \geq \underline{\mathbf{a}}^{(\mathbf{final}),\top}\mathbf{x} + \underline{\mathbf{c}}^{(\mathbf{final})} =: \underline{\mathbf{f}}(\mathbf{x})$$

$$\underline{\mathbf{A}}^{(\mathbf{i})}\mathbf{x} + \underline{\mathbf{c}}^{(\mathbf{i})} \leq \mathbf{z}^{(\mathbf{i})}(\mathbf{x}) \leq \overline{\mathbf{A}}^{(\mathbf{i})}\mathbf{x} + \overline{\mathbf{c}}^{(\mathbf{i})}$$

Our Core Insight: Don't Waste Information

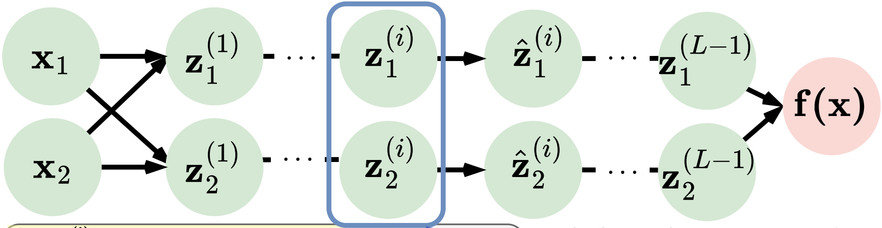
- Linear bound propagation provides linear relaxations at every layer.
- Only the final layer's relaxation is used to determine if a subproblem has been verified.



$$\mathbf{f}(\mathbf{x}) \geq \underline{\mathbf{a}}^{(\text{final}),\top}\mathbf{x} + \underline{\mathbf{c}}^{(\text{final})} =: \underline{\mathbf{f}}(\mathbf{x})$$

Our Core Insight: Don't Waste Information

- Linear bound propagation provides linear relaxations at every layer.
- Only the final layer's relaxation is used to determine if a subproblem has been verified.
- Even the final layer's relaxation is **discarded** if the subproblem cannot be verified before branching further.
- Why not **re-purpose** them as constraints to strengthen the relaxation?



$\max_{\mathbf{x}\in\mathcal{S}} \mathbf{z}_\mathbf{j}^{(\mathbf{i})}$, **such that** $\mathbf{Gx}+\mathbf{h}\leq\mathbf{0}$

where $\mathbf{Gx}+\mathbf{h}$ are a set of linear constraints,
e.g. split constraints or output constraint $\underline{\mathbf{f}}(\mathbf{x})\leq\mathbf{0}$

Tightening intermediate bounds **benefit** the final relaxation, $\underline{\mathbf{f}}(\mathbf{x})$

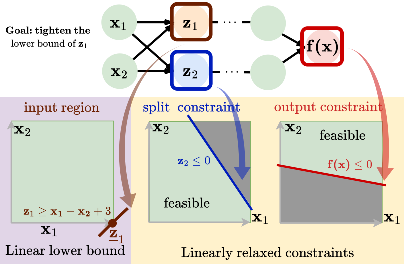
What are the constraints? Where do they come from?

Setting. Input box $X = \{x \mid \hat{x} - \epsilon \le x \le \hat{x} + \epsilon\}$.
Source A - Final-layer (output) constraints.

$$f(x) \ge a^{(final),\top} x + c^{(final)} > 0 \;\Rightarrow\; a^{(final),\top} x + c^{(final)} \le 0.$$

Source B - Activation-split constraints.

$$z_j^{(i)} \ge s \Rightarrow \overline{A}_j^{(i)\top} x + \overline{c}_j^{(i)} \ge s,$$
$$z_j^{(i)} \le s \Rightarrow \underline{A}_j^{(i)\top} x + \underline{c}_j^{(i)} \le s.$$



Constraints arise from BaB splits / properties.

These affine inequalities are *free byproducts*:

- tighten per-neuron bounds (Complete Clipping, 1D dual)
- shrink the input box (Relaxed Clipping, closed form)

Primal Formulation for Bound Tightening

Direct Refinement

$$L^\star = \min_{\boldsymbol{x}\in\mathcal{S}} \{ \boldsymbol{a}^\top \boldsymbol{x} + c : \boldsymbol{g}^\top \boldsymbol{x} + h \le 0 \} \qquad (2)$$

Per-Dimension Refinement

$$\begin{aligned} \underline{\boldsymbol{x}}_i &= \min_{\boldsymbol{x}\in\mathcal{S}} \{ \boldsymbol{x}_i \mid \boldsymbol{A}\boldsymbol{x} + \boldsymbol{c} \le 0 \}, \\ \bar{\boldsymbol{x}}_i &= \max_{\boldsymbol{x}\in\mathcal{S}} \{ \boldsymbol{x}_i \mid \boldsymbol{A}\boldsymbol{x} + \boldsymbol{c} \le 0 \}. \end{aligned} \qquad (3)$$

- It is overwhelming to refine the lower and upper bounds of *every* neuron using Eq. (2).
    - A vision transformer with ∼76k parameters amounts to solving Eq. (2) 152k times!
- Likewise, for an n-dimensional input, Eq. (3) must be solved $2n$ times.
- This is for a **single** subproblem in branch-and-bound. This is completely **intractable** for LP solvers.
- This motivates the need for an **effective** and **scalable** manner to produce such bounds.

Our Algorithms: Two Efficient Ways to Clip

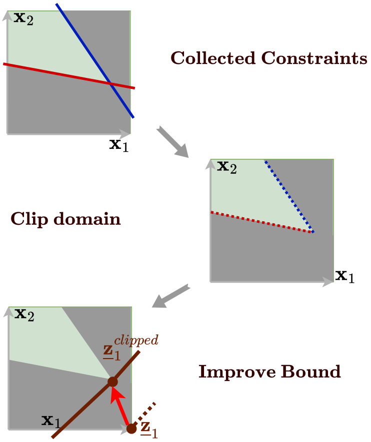
Complete Clipping:

- Tightens bounds for *specific* critical neurons.
- Solve a **1D Lagrangian dual** (not an n-D LP).
- Exact, GPU-friendly implementation.

Theorem 3.1

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

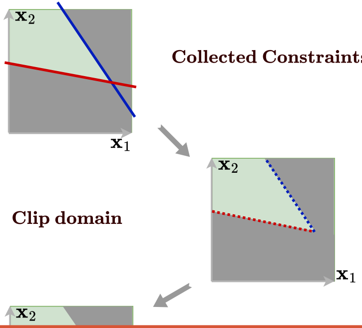
Dual $\beta \in \mathbb{R}_+$, affine relaxation, $\boldsymbol{a} \in \mathbb{R}^n$, $c \in \mathbb{R}$, and linear constraint, $\boldsymbol{g}^\top \boldsymbol{x} + h \leq 0$, $\boldsymbol{g} \in \mathbb{R}^n$, $h \in \mathbb{R}$.



Our Algorithms: Two Efficient Ways to Clip

Complete Clipping:

- Tightens bounds for *specific* critical neurons.
- Solve a **1D Lagrangian dual** (not an n-D LP).
- Exact, GPU-friendly implementation.



Theorem 3.1

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

Dual $\beta \in \mathbb{R}_+$, affine relaxation, $\boldsymbol{a} \in \mathbb{R}^n$, $c \in \mathbb{R}$, and linear constraint, $\boldsymbol{g}^\top \boldsymbol{x} + h \leq 0$, $\boldsymbol{g} \in \mathbb{R}^n$, $h \in \mathbb{R}$.

Compared with LP dual solver

Specialized Complete Clipping is $\sim$**880**$\times$ **faster** than Gurobi, a standard LP solver (comparable accuracy).

Our Algorithms: Two Efficient Ways to Clip

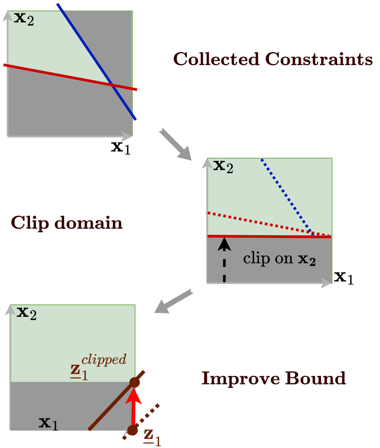
Relaxed Clipping:

- Tightens bounds for *all* neurons at once by refining the input box.
- **Closed-form** $O(n)$ update (no solver).

Theorem 3.2

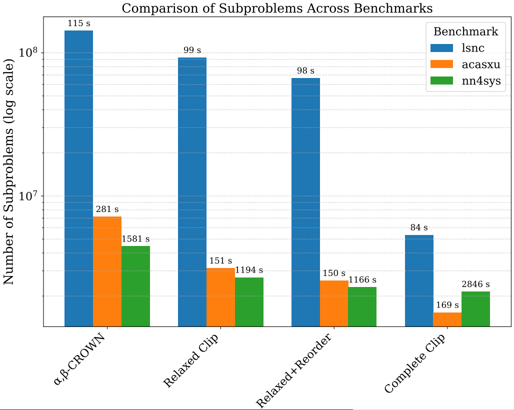
$$\begin{cases} \overline{\boldsymbol{x}}_i^{(\text{new})} = \min\left\{\boldsymbol{x}_i^{(\text{clip})}, \overline{\boldsymbol{x}}_i\right\} & \text{if } \boldsymbol{a}_i > 0 \\ \underline{\boldsymbol{x}}_i^{(\text{new})} = \max\left\{\boldsymbol{x}_i^{(\text{clip})}, \underline{\boldsymbol{x}}_i\right\}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

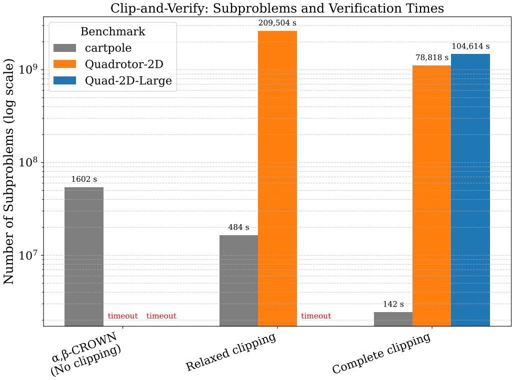
With constraint, $\boldsymbol{a}^\top \boldsymbol{x} + c \le 0$, $\boldsymbol{a} \in \mathbb{R}^n$, $\boldsymbol{c} \in \mathbb{R}$, and $\boldsymbol{x}_i^{(\text{clip})} = \left(-\sum_{j \neq i}\{\boldsymbol{a}_j \hat{\boldsymbol{x}}_j - |\boldsymbol{a}_j|\epsilon_j\} - c\right)/\boldsymbol{a}_i$.



Applications: Does It Work? Yes.

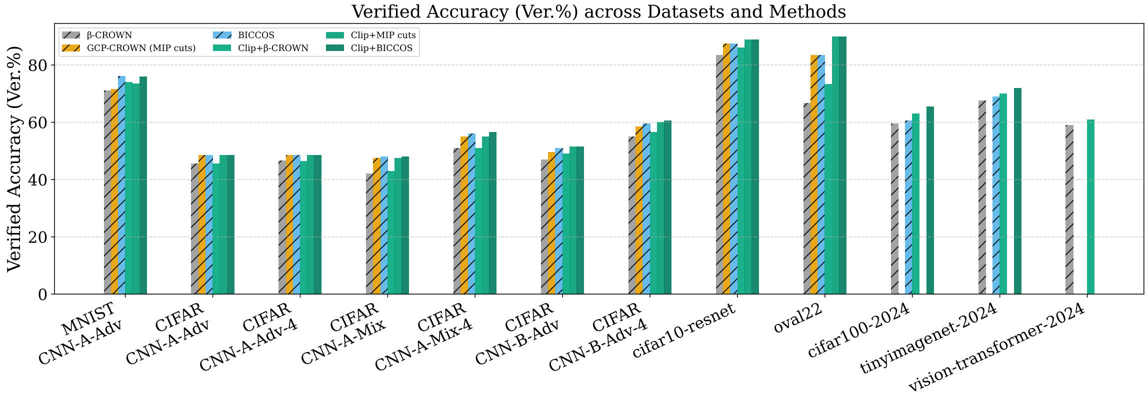
- **Massive speedup:** Up to **96%** fewer BaB subproblems on `lsnc`.
- **Hard problems:** Baselines time out on control tasks (e.g., `Quadrotor-2D`); Clip-and-Verify finishes.





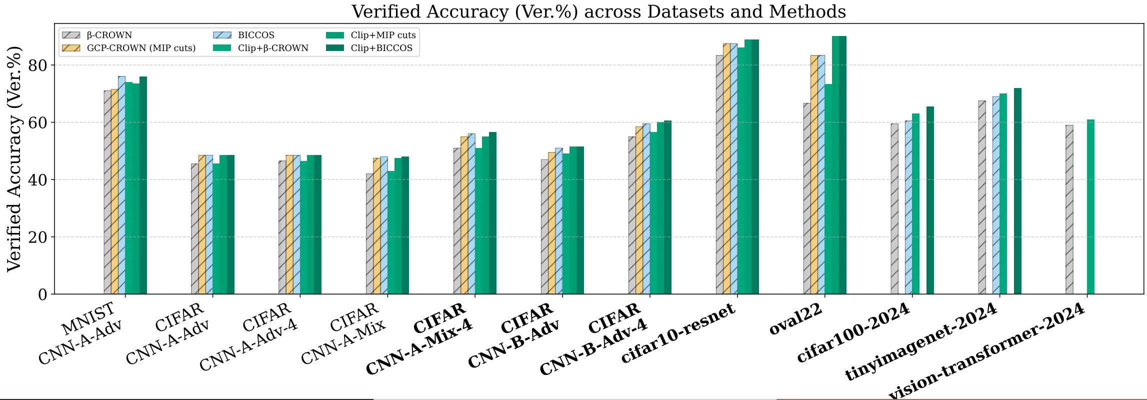
Applications: Does It Work? Yes.

- **Support all net structures:** We support any activation functions including MatMul, ReLU, Sigmoid, and Softmax.
- **SOTA:** Integrated into a VNN-COMP 2025 winning stack.



Applications: Does It Work? Yes.

- **Support all net structures:** We support any activation functions including MatMul, ReLU, Sigmoid, and Softmax.
- **SOTA:** Integrated into a VNN-COMP 2025 winning stack.



Conclusion

- **Insight:** Re-purpose **free linear constraints** that others discard.
- **Method:** Two GPU-friendly algorithms:
  - **Complete Clipping**: precise, $\sim 880\times$ faster dual solver.
  - **Relaxed Clipping**: fast, global box shrink.
- **Impact:** More properties, faster; enables previously intractable systems.
- **Code:** github.com/Verified-Intelligence/Clip_and_Verify

Thank you!

Clip-and-Verify is integrated into the state-of-the-art verifier α, β-CROWN, the overall winner of the International Verification of Neural Networks Competition (VNN–COMP) 2021–2025.