OUT OF THE SHADOWS: EXPLORING A LATENT SPACE FOR NEURAL NETWORK VERIFICATION

Anonymous authors

000

001

002 003 004

010 011

012

013

014

016

017

018

019

021

025

026

027

028

029

031

032

034

037

040

041

042

043

044

046

047

048

051

052

Paper under double-blind review

ABSTRACT

Neural networks are ubiquitous. However, they are often sensitive to small input changes. Hence, to prevent unexpected behavior in safety-critical applications, their formal verification – a notoriously hard problem – is necessary. Many stateof-the-art verification algorithms use reachability analysis or abstract interpretation to enclose the set of possible outputs of a neural network. Often, the verification is inconclusive due to the conservatism of the enclosure. To address this problem, we design a novel latent space for formal verification that enables the transfer of output specifications to the input space for an iterative specificationdriven input refinement, i.e., we iteratively reduce the set of possible inputs to only enclose the unsafe ones. The latent space is constructed from a novel view of projection-based set representations, e.g., zonotopes, which are commonly used in reachability analysis of neural networks. A projection-based set representation is a "shadow" of a higher-dimensional set – a latent space – that does not change during a set propagation through a neural network. Hence, the input set and the output enclosure are "shadows" of the same latent space that we can use to transfer constraints. We present an efficient verification tool for neural networks that uses our iterative refinement to significantly reduce the number of subproblems in a branch-and-bound procedure. Using zonotopes as a set representation, unlike many other state-of-the-art approaches, our approach can be realized by only using matrix operations, which enables a significant speed-up through efficient GPU acceleration. We demonstrate that our tool achieves competitive performance, which would place it among the top-ranking tools of the last neural network verification competition (VNN-COMP'24).

1 Introduction

Neural networks perform exceptionally well across many complex tasks, e.g., object detection Zou et al. (2023) or protein structure prediction Jumper et al. (2021). However, neural networks can be sensitive towards small input changes, e.g., often adversarial attacks can provoke misclassifications Goodfellow et al. (2015). Thus, neural networks must be formally verified to avoid unexpected behavior in safety-critical applications, e.g., autonomous driving Chib & Singh (2023) or airborne collision avoidance Irfan et al. (2020), where the inputs can be influenced by sensor noise or external disturbances.

The goal of formal verification of neural networks is to find a mathematical proof that every possible output for a given input set is safe with respect to a given specification; in this work, we demand that the output avoids an unsafe set. The verification problem is undecidable in the general case Ivanov et al. (2019) and NP-complete for neural networks with ReLU-activiation functions (Katz et al., 2017, Appendix I). Many prominent verification algorithms use reachability analysis or abstract interpretation to enclose the intersection of an unsafe set with the output of the neural network Gehr et al. (2018); Singh et al. (2019a); Xu et al. (2021); Bak (2021); Kochdumper et al. (2023): The input set is represented using a continuous set representation (such as intervals or zonotopes), which is conservatively propagated through the layers of a neural network to enclose all possible outputs. If the intersection of the output enclosure with an unsafe set is empty, the safety of the given input set is formally verified. Out of the box, most reachability-based algorithms do not scale well to large neural networks with high-dimensional input spaces because the conservatism of the set propagation increases due to over-approximating nonlinearities in the neural network. Most verification

055

056

057

058

060

061

062

063

064

065

066

067

068

069

070

071

072

073

074

075

076

077 078

079

081

082

084

086

087

090

091 092 093

094

096

098

099

100

101

102

103

104

105

106

107

algorithms reduce the conservatism by integrating a branch-and-bound procedure to recursively split the verification problem into smaller and simpler subproblems, e.g., by exploiting the piecewise linearity of a ReLU-activiation function to reduce approximation errors Bunel et al. (2018) or splitting the input set to reduce its size. However, in the worst case, the verification problem is split into exponentially many subproblems, all of which must be verified.

In this paper, we speed up the formal verification of neural networks by iteratively refining the input set to only enclose the unsafe inputs. Thereby, we reduce the size of the input set to reduce the number of splits and, ultimately, the number of subproblems to be verified. For our iterative input refinement, we construct a novel latent space to transfer the unsafe set backwards through the network from the output space to the input space for the enclosure of all unsafe inputs, i.e., we can discard all inputs that are already proven to be safe. We construct the latent space by using projection-based set representations that represent the projection ("shadow") of a higher-dimensional set, e.g., a zonotope is the "shadow" of a higher-dimensional hypercube. The set propagation through a neural network only changes the "shadow", while the higher-dimensional set remains unchanged. Hence, all considered sets are different "shadows" of the same higher-dimensional set, representing a latent space. We can exploit the dependencies between the considered sets through our novel latent space to transfer the unsafe set from the output space to the input space to refine the input set Kochdumper et al. (2020). Moreover, if we cannot verify the safety of the neural network, we can utilize the latent space to extract candidates for counterexamples. Ultimately, we propose a verification algorithm that integrates our novel iterative input refinement into a branch-and-bound procedure for verifying and falsifying neural networks. Further, unlike many other state-of-the-art verification algorithms, we implement our verification algorithm using only matrix operations to take full advantage of GPU acceleration to simultaneously verify entire batches of subproblems.

To summarize, our main contributions are:

- A latent space for the verification of neural networks constructed from a novel view of the propagation of projection-based sets, where all considered sets are "shadows" of the same higher-dimensional set.
- By utilizing the latent space, we propose an iterative specification-driven input refinement and an approach for counterexample extraction to speed up branch-and-bound verification and falsification of neural networks.
- An efficient algorithm for neural network verification that takes full advantage of GPU acceleration.
- An extensive evaluation and comparison with state-of-the-art neural network verification algorithms on benchmarks from the neural network verification competition 2024 (VNN-COMP'24) Brix et al. (2024). Additionally, we conduct extensive ablation studies to justify the design choices of our verification algorithm.

2 PRELIMINARIES

2.1 NOTATION

Lowercase letters denote vectors and uppercase letters denote matrices. We denote the i-th entry of a vector x by $x_{(i)}$ and the entry in the i-th row and the j-th column of a matrix $A \in \mathbb{R}^{n \times m}$ by $A_{(i,j)}$. The i-th row is written as $A_{(i,\cdot)}$ and the j-th column as $A_{(\cdot,j)}$. The identity matrix is denoted by $I_n \in \mathbb{R}^{n \times n}$. We write the matrix (with appropriate size) that contains only zeros or ones as $\mathbf{0}$ and $\mathbf{1}$. Given two matrices $A \in \mathbb{R}^{m \times n_1}$ and $B \in \mathbb{R}^{m \times n_2}$, their (horizontal) concatenation is denoted by $[A B] \in \mathbb{R}^{m \times (n_1 + n_2)}$. The operation Diag(v) returns a diagonal matrix with the entries of vector v on its diagonal, and the operation |A| takes the elementwise absolute value of a matrix $A \in \mathbb{R}^{n \times m}$. We denote sets by uppercase calligraphic letters. Given two sets $S_1, S_2 \subset \mathbb{R}^n$, we write their Minkowski sum as $S_1 \oplus S_2 = \{s_1 + s_2 \mid s_1 \in S_1, s_2 \in S_2\}$. For $n \in \mathbb{N}$, $[n] = \{1, 2, \ldots, n\}$ is the set of all natural numbers up to n. The n-dimensional interval $\mathcal{I} \subset \mathbb{R}^n$ with bounds $l, u \in \mathbb{R}^n$ is written as $\mathcal{I} = [\underline{x}, \overline{x}]$, where $\forall i \in [n] : \underline{x}_{(i)} \leq \overline{x}_{(i)}$. For a function $f : \mathbb{R}^n \to \mathbb{R}^m$, we abbreviate the image of $S \subset \mathbb{R}^n$ with $f(S) = \{f(s) \mid s \in S\}$.

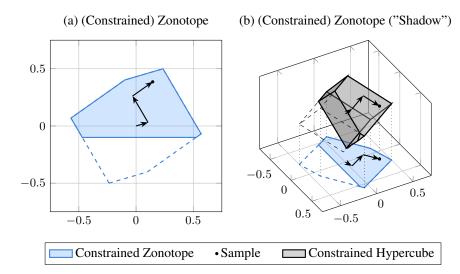


Figure 1: (a) An illustration of a constrained zonotope and a sample with its generators; (b) the same constrained zonotope as the "shadow" of a constrained hypercube.

2.2 FEED-FORWARD NEURAL NETWORKS

A (feed-forward) neural network $\Phi \colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_\kappa}$ is a sequence of $\kappa \in \mathbb{N}$ layers. Each layer applies an affine map (linear layer) or an element-wise nonlinear activation function (nonlinear layer).

Definition 1 (Neural Network, (Bishop, 2006, Sec. 5.1)). For an input $x \in \mathbb{R}^{n_0}$, the output of a neural network $y = \Phi(x) \in \mathbb{R}^{n_{\kappa}}$ is

$$h_0 = x,$$
 $h_k = L_k(h_{k-1})$ for $k \in [\kappa]$, $y = h_{\kappa}$,

where

$$L_k(h_{k-1}) = \begin{cases} W_k \, h_{k-1} + b_k & \textit{if k-th layer is linear,} \\ \phi_k(h_{k-1}) & \textit{otherwise,} \end{cases}$$

with weights $W_k \in \mathbb{R}^{n_k \times n_{k-1}}$, bias $b_k \in \mathbb{R}^{n_k}$, and (element-wise) nonlinear function ϕ_k .

2.3 SET-BASED COMPUTING

A zonotope is a convex set representation popular in reachability analysis due to its favorable computational complexity Singh et al. (2019b); Kochdumper et al. (2023).

Definition 2 (Zonotope, (Girard, 2005, Def. 1)). Given a center $c \in \mathbb{R}^n$ and a generator matrix $G \in \mathbb{R}^{n \times q}$, a zonotope is defined as

$$\mathcal{Z} = \langle c, G \rangle_Z := \left\{ c + \sum_{i=1}^q G_{(\cdot, i)} \beta_{(i)} \middle| \beta \in [-1, 1] \right\}.$$

Subsequently, we define the set-based operations for zonotopes required for our verification algorithm. The Minkowski sum of a zonotope $\mathcal{Z}=\langle c,G\rangle_Z\subset\mathbb{R}^n$ and an interval $[\underline{x},\overline{x}]\subset\mathbb{R}^n$ is computed by (Althoff, 2010, Prop. 2.1 & Sec. 2.4)

$$\mathcal{Z} \oplus [\underline{x}, \overline{x}] = \langle c + 1/2 (\overline{x} + \underline{x}), [G \operatorname{diag}(1/2 (\overline{x} - \underline{x}))] \rangle_{Z}. \tag{1}$$

The image of a zonotope $\mathcal{Z} = \langle c, G \rangle_Z \subset \mathbb{R}^n$ under an affine map $f \colon \mathbb{R}^n \to \mathbb{R}^m$, $x \mapsto W x + b$ with $W \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ is computed by (Althoff, 2010, Sec. 2.4)

$$f(\mathcal{Z}) = W \mathcal{Z} \oplus b = \langle W c + b, W G \rangle_{Z}. \tag{2}$$

Using an affine map, we can write a zonotope $\mathcal{Z}=\langle c,G\rangle_Z\subset\mathbb{R}^n$ with q generators, i.e., $G\in\mathbb{R}^{n\times q}$, as the projection of a q-dimensional (unit)-hypercube $\mathcal{B}_q=[-1,1]\colon\mathcal{Z}=c\oplus G\,\mathcal{B}_q$. Thus, intuitively, a zonotope is the "shadow" of a higher-dimensional hypercube.

A convex polytope is the intersection of a finite number of halfspaces (Althoff, 2010, Def. 2.1); we denote a polytope by $\langle A,b\rangle_H=\{x\in\mathbb{R}^n\mid A\,x\leq b\}\subset\mathbb{R}^n$, where $A\in\mathbb{R}^{n\times p}$ and $b\in\mathbb{R}^p$.

All zonotopes are convex and point-symmetric. However, by constraining the hypercube, we can represent arbitrary convex polytopes (Scott et al., 2016, Thm. 1).

Definition 3 (Constrained Zonotope, (Scott et al., 2016, Def. 3)). Given a zonotope $\mathcal{Z} = \langle c, G \rangle_Z \subset \mathbb{R}^n$ with $c \in \mathbb{R}^n$ and $G \in \mathbb{R}^{n \times q}$, the zonotope with constraints $C \beta \leq d$, for $\beta \in \mathcal{B}_q$ with $C \in \mathbb{R}^{p \times q}$ and $d \in \mathbb{R}^p$, is defined as

$$\mathcal{Z}|_{C < d} := \{c + G\beta \mid \beta \in [-1, 1], C\beta \leq d\} = c \oplus G\overline{\mathcal{B}_q},$$

where $\overline{\mathcal{B}_q} := \{\beta \in [-1,1] \mid C \beta \leq d\} = \mathcal{B}_q \cap \langle C, d \rangle_H$ is the constrained unit-hypercube.

Compared to (Scott et al., 2016, Def. 3), we use inequality constraints instead of equality constraints for convenience. Both types of constraints are equivalent and can be translated by introducing slack variables. Moreover, the considered set-based operations, i.e., Minkowski sum (1) and affine map (2), are identical for zonotopes and constrained zonotope (Scott et al., 2016, Prop. 1). Fig. 1 illustrates a constrained zonotope as a "shadow" of a constrained hypercube.

2.4 FORMAL VERIFICATION OF NEURAL NETWORKS

The output set of a neural network can be enclosed by conservatively propagating a zonotope through the layers of the neural network.

Proposition 1 (Set Propagation, (Ladner & Althoff, 2023, Sec. 2.4)). Given a neural network $\Phi \colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_\kappa}$ and an input set $\mathcal{X} \subset \mathbb{R}^{n_0}$, an enclosure $\mathcal{Y} = \text{enclose}(\Phi, \mathcal{X}) \subset \mathbb{R}^{n_\kappa}$ of the image $\mathcal{Y}^* := \Phi(\mathcal{X}) \subseteq \mathcal{Y}$ can be computed as

$$\mathcal{H}_0 = \mathcal{X}, \hspace{1cm} \mathcal{H}_k = \mathtt{enclose}(L_k, \mathcal{H}_{k-1}) \hspace{0.2cm} \mathit{for} \hspace{0.05cm} k \in [\kappa], \hspace{1cm} \mathcal{Y} = \mathcal{H}_{\kappa}.$$

The operation $\operatorname{enclose}(L_k,\mathcal{H}_{k-1})$ encloses the image of the k-th layer for the input set \mathcal{H}_{k-1} , i.e., $L_k(\mathcal{H}_{k-1})\subseteq\operatorname{enclose}(L_k,\mathcal{H}_{k-1})$ (Ladner & Althoff, 2023, Prop. 2.14). If the k-th layer is linear, an affine map is applied (2): $\operatorname{enclose}(L_k,\mathcal{H}_{k-1})=W_k\,\mathcal{H}_{k-1}\oplus b_k$; otherwise, the activation function ϕ_k is enclosed with a linear function and $\operatorname{enclose}(L_k,\mathcal{H}_{k-1})=\operatorname{diag}(m_k)\,\mathcal{H}_{k-1}\oplus [\underline{e}_k,\overline{e}_k]$, where $m_{k(i)}$ is the approximation slope and $[\underline{e}_{k(i)},\overline{e}_{k(i)}]$ the approximation error of the i-th neuron, with $i\in[n_k]$ (Koller et al., 2025, Sec. IV).

2.5 PROBLEM STATEMENT

Given a neural network $\Phi \colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_\kappa}$, an input set $\mathcal{X} \subset \mathbb{R}^{n_0}$, and an unsafe set $\mathcal{U} \subset \mathbb{R}^{n_\kappa}$, our goal is to derive an efficient and practical algorithm that can either formally verify the safety of the neural network, i.e.,

$$\Phi(\mathcal{X}) \cap \mathcal{U} = \emptyset, \tag{3}$$

or find a counterexample, i.e.,

$$x \in \mathcal{X}$$
 such that $\Phi(x) \in \mathcal{U}$. (4)

3 A NOVEL VIEW ON ZONOTOPE PROPAGATION

A zonotope is the "shadow" of a higher-dimensional hypercube (Fig. 1b). The propagation of a zonotope through the layers of a neural network transforms the projection of the hypercube, but the hypercube itself remains unchanged. Thus, all enclosed sets, i.e., the input, hidden, and output sets, are different "shadows" of the same hypercube. Therefore, the hypercube represents a latent space. Let us demonstrate this novel view by an example:

Example 1. Fig. 2 illustrates the propagation of a two-dimensional input set \mathcal{X} through a linear layer and a ReLU-activation function, i.e., $x \mapsto \operatorname{ReLU}(2^{-1/2} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} x + \begin{bmatrix} 1 \\ 0 \end{bmatrix})$. Intuitively, the linear layer rotates the hypercube, and the nonlinear layer tilts it to compute the output set as its "shadow."

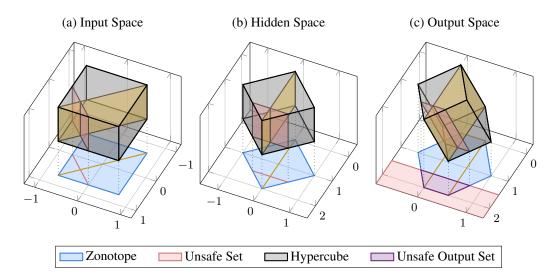


Figure 2: Illustrating the zonotope propagation through a linear layer and a nonlinear layer as the "shadows" of the same hypercube: (a) The input set in the input space; (b) The output of the linear layer, which rotates and offsets the input set; (c) The output of the nonlinear layer tilts the hypercube to add the approximation errors. Using the hypercube, the unsafe set () is transferred from the output space to the input space. Additionally, the hypercube is split along a hyperplane () to exploit the piecewise linearity of the ReLU-activation function.

For input set $\mathcal{X} = \langle \mathbf{0}, 2^{-1/2} I_2 \rangle_{\mathcal{Z}}$, the hidden set \mathcal{H}_1 and the \mathcal{Y} are computed using Prop. 1:

$$\mathcal{H}_{1} = 2^{-1/2} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \mathcal{X} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \left\langle \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \frac{1}{2} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \right\rangle_{\mathcal{Z}},$$

$$\mathcal{Y} = \operatorname{diag}\left(\begin{bmatrix} 1 \\ \frac{1}{2} \end{bmatrix} \right) \mathcal{H}_{1} + \left[\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \frac{1}{2} \end{bmatrix} \right] = \left\langle \begin{bmatrix} 1 \\ \frac{1}{4} \end{bmatrix}, \frac{1}{4} \begin{bmatrix} 2 & -2 & 0 \\ 1 & 1 & 1 \end{bmatrix} \right\rangle_{\mathcal{Z}}$$

$$(5)$$

The approximation slope and errors for enclosing the output of the ReLU-activation function are computed using (Koller et al., 2025, Prop. 10). For each input $x \in \mathcal{X}$ and its corresponding output $y = \Phi(x) \in \mathcal{Y}$, we use the definition of a zonotope (Def. 2) to obtain $\beta \in [-1,1]^3$ such that

$$x = \begin{bmatrix} 1/\sqrt{2} & 0\\ 0 & 1/\sqrt{2} \end{bmatrix} \beta_{([2])}, \qquad y = \begin{bmatrix} 1\\ 1/4 \end{bmatrix} + \begin{bmatrix} 1/2 & -1/2\\ 1/4 & 1/4 \end{bmatrix} \beta_{([2])} + \begin{bmatrix} 0\\ 1/4 \end{bmatrix} \beta_{(3)}.$$
 (6)

The input x and the output y are represented using the same factors $\beta_{([2])}$ with an additional factor $\beta_{(3)}$ for the approximation error. We can use an unsafe set $y_{(1)} \geq 3/2$ to formluate constraints (\square in Fig. 2) on the factors $\beta_{([2])}$ of the input:

$$y_{(1)} \ge 3/2 \iff 1 + \begin{bmatrix} 1/2 & -1/2 \end{bmatrix} \beta_{([2])} + 0 \beta_{(3)} \ge 3/2 \iff \underbrace{\begin{bmatrix} -1/2 & 1/2 \end{bmatrix}}_{=C} \beta_{([2])} \le \underbrace{-1/2}_{=d}.$$
 (7)

Therefore, all unsafe inputs are enclosed by $\mathcal{X}|_{C\leq d}$. Analogously, we can split the set along $h_1\leq 0\iff [1/2\quad 1/2]\ \beta_{([2])}\leq 0$, for $h_1\in\mathcal{H}_1$, to exploit the piecewise linearity of the ReLU-activiation function (\square in Fig. 2).

As example 1 demonstrates, we can contrain the input set with an unsafe set in the output space by exploiting the dependencies between the considered sets through the latent space. Prop. 2 formalizes an input refinement based on our observations.

Proposition 2 (Enclosing Unsafe Inputs). Given are a neural network $\Phi \colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_\kappa}$, an input set $\mathcal{X} = \langle c_x, G_x \rangle_Z \subset \mathbb{R}^{n_0}$ with $G_x \in \mathbb{R}^{n_0 \times q_0}$, and an unsafe set $\mathcal{U} = \langle A, b \rangle_H \subset \mathbb{R}^{n_\kappa}$. Let $\mathcal{Y} = \langle c_y, G_y \rangle_Z = \text{enclose}(\Phi, \mathcal{X})$ be an enclosure of the output set with $G_y \in \mathbb{R}^{n_\kappa \times q_\kappa}$. We enclose all unsafe inputs with

$$\{x \in \mathcal{X} \mid \Phi(x) \in \mathcal{U}\} \subseteq \mathcal{X}|_{C \le d}$$

where $C := A G_{y(\cdot,[q_0])}$ and $d := b - A c_y + \left| A G_{y(\cdot,[q_\kappa]\setminus[q_0])} \right| \mathbf{1}$.

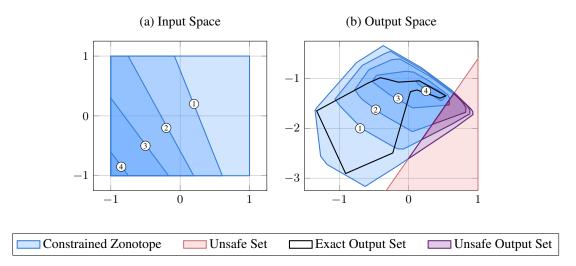


Figure 3: Illustration of an iterative refinement of the input set: The input set is iteratively refined (Prop. 2). After the fourth iteration, the intersection with the unsafe set is empty, and the neural network is verified.

Proof. See Appendix D.

We can iteratively apply Prop. 2 to refine the input set to minimize the intersection of the computed output set and the unsafe set. During each iteration, we transfer the unsafe set from the output space to the input space to remove the parts of the input set that are provably safe. In Fig. 3, we iterate Prop. 2 until the intersection of the output set and the unsafe set is empty, and thus the problem is verified.

4 SET-BASED FALSIFICATION, VERIFICATION, AND INPUT REFINEMENT

For the fast and practical verification of neural networks, we integrate our novel input refinement (Prop. 2) into a branch-and-bound procedure (Alg. 1), where we utilize the enclosed output set for verification, falsification, and input refinement. In each iteration, we perform the following computations: (i) The output set of the current input set is enclosed (Prop. 1). (ii) The intersection with the unsafe set is checked (Sec. 4.1). (iii) If the intersection is non-empty, the verification is inconclusive, and falsification is attempted (Sec. 4.2). (iv) Finally, if we cannot verify the input set nor find a counterexample, we refine the input set (Prop. 2). Moreover, we reduce the conservatism of the zonotope propagation by splitting the refined input set through additional constraints of the hypercube (Scott et al., 2016, Prop. 3); e.g., we can formulate constraints that exploit the piecewise linearity of the ReLU-activation function (in Fig. 2). Please see Appendix B for details on our splitting heuristic. The subsequent subsections describe the steps of Alg. 1 in detail.

4.1 SET-BASED VERIFICATION

We compute the output set $\mathcal{Y}=\langle c_y,G_y\rangle_Z\subset\mathbb{R}^{n_\kappa}$ of the current input set using Prop. 1 and check if the intersection with the unsafe set $\mathcal{U}=\langle A,b\rangle_H\subset\mathbb{R}^{n_\kappa}$ with $A\in\mathbb{R}^{p\times n_\kappa}$ is empty (Scott et al., 2016, Prop. 2):

$$\exists i \in [p] \colon A_{(i,\cdot)} c_y - |A_{(i,\cdot)} G_y| \mathbf{1} > b_{(i)} \implies \mathcal{Y} \cap \mathcal{U} = \emptyset.$$
 (8)

4.2 Set-Based Falsification

If we cannot verify the current input set, we try to find a counterexample within it. For that, we utilize the latent space (Sec. 3) to identify the input for a boundary point of the intersection of the enclosed output set and the unsafe set. For each normal vector of the unsafe set specification $A_{(i,\cdot)} \in \mathbb{R}^{n_{\kappa}}$,

325

326

327

328

330

331

332

333

334

335

336

337

338

340

341

342

343

344

345

346

347

348

349

350

351 352 353

354

355

356 357

358 359

360 361

362

364

365

366

367

368

369

370371

372 373

374375

376

377

18

19

20

21

```
Algorithm 1: Set-based verification algorithm. We store input sets in a queue and use the oper-
   ations initQueue, isNonEmpty, enqueue, and dequeue from (Knuth, 1997, Sec. 2.2.1)
   Data: Neural network \Phi \colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_\kappa}, input set \mathcal{X} = \langle c_x, G_x \rangle_Z \subset \mathbb{R}^{n_0}, unsafe set
            \mathcal{U} = \langle A, b \rangle_H \subset \mathbb{R}^{n_\kappa}
   Result: Verification result: VERIFIED, FALSIFIED(\tilde{x}) with counterexample \tilde{x} \in \mathcal{X}
1 function verify(\Phi, \mathcal{X}, \mathcal{U})
       Q \leftarrow initQueue(), i \leftarrow 1
2
                                                                                             // Initialize queue & counter.
       Q.enqueue(\mathcal{X})
                                                                                              // Add the initial input set.
3
       while Q.isNonEmpty() do
4
          \mathcal{X}_i \leftarrow \mathsf{Q}.\mathsf{dequeue}()
5
                                                                                                   // Get the next input set.
          \mathcal{Y}_i \leftarrow \mathtt{enclose}(\Phi, \mathcal{X}_i)
                                                                                    // Enclose the output set (Prop. 1).
6
          if \mathcal{Y}_i \cap \mathcal{U} = \emptyset then
7
                                                                                             // 1. Verification (Sec. 4.1)
              continue
8
                                                                                                        // Verified input set.
          else
                                                                                            // 2. Falsification (Sec. 4.2)
9
              \tilde{x}_i \leftarrow \text{Compute an adversarial input.}
10
              \tilde{y}_i \leftarrow \Phi(x)
11
                                                                          // Compute the adversarial output (Def. 1).
              if \tilde{y}_i \in \mathcal{U} then
12
                 return FALSIFIED(\tilde{x}_i)
                                                                                                    // Found an unsafe input.
13
                                                              // 3. Refinement & Splitting (Prop. 2, Appendix B)
14
                  C, d \leftarrow \text{Compute the constraints for the input set.}
                                                                                                                          // Prop. 2
15
                 \mathcal{X}_i|_{C_1 \leq d_1}, \dots, \mathcal{X}_i|_{C_{\xi} \leq d_{\xi}} \leftarrow \text{Refine and split the input set.}
                                                                                                           // Prop. 2, (Scott
16
                   et al., 2016, Prop. 3)
                 Q.enqueue (\mathcal{X}_i|_{C_1 < d_1}, \dots, \mathcal{X}_i|_{C_{\varepsilon} < d_{\varepsilon}})
17
                                                                                             // Add new sets to the queue.
```

for $i \in [p]$, we compute a boundary point (Althoff & Frehse, 2016, Lemma 1):

$$\tilde{\beta}_i = \operatorname{sign}(A_{(i,\cdot)} G_y), \qquad \qquad \tilde{y} = c_y + G_y \,\tilde{\beta}_i. \tag{9}$$

// Increment the counter.

// Queue is empty; verified all input sets.

Finally, we check if the corresponding input $\tilde{x} = c_x + G_x \, \tilde{\beta}_{([q_0])} \in \mathcal{X}$ in the input set $\mathcal{X} = \langle c_x, G_x \rangle_Z$ with $G_x \in \mathbb{R}^{n_0 \times q_0}$ is a counterexample, i.e., whether $\Phi(\tilde{x}) \in \mathcal{U}$.

5 EVALUATION

 $i \leftarrow i + 1$

end

return VERIFIED

end

end

We have implemented our verification algorithm in MATLAB. To make our evaluation as transparent and reproducible as possible, we compare the results of our tool on benchmarks from the neural network verification competition 2024 (VNN-COMP'24) Brix et al. (2024) with the top-5 tools of last year's competition: α - β -CROWN Wang et al. (2021), NeuralSAT Duong et al. (2023), PyRat Lemesle et al. (2024), Marabou Wu et al. (2024), and nnenum Bak (2021).

Tab. 1 presents the results for 8 competitive and standardized benchmarks of the VNN-COMP'24 Brix et al. (2024). Please see Appendix A for details. Across all benchmarks, our verification algorithm achieves competitive performance, even matching the top performance in four benchmarks. These results would place our tool in the top 5 of the competition.

5.1 ABLATION STUDIES

We run extensive ablation studies to justify the different design choices of our algorithm.

Input Refinement We evaluate the improvement through our input refinement (Tab. 2a). Therefore, we compare the number of subproblems required for verifying instances with (\checkmark) and without (\checkmark) our input refinement enabled: We observe that our refinement reduces the number of subproblems by 60.5% for acasxu and 24.8% for safenlp.

Table 1: Main Results [%Solved \uparrow (#Verified / #Falsified)].

Tool	acasxu	collins -rul-cnn	cora	dist -shift	linear -izenn	meta -room	safenlp	tllverify -bench
Ours	99.5% (138 / 47)	100.0 % (30 / 32)	81.1% (18 / 128)	98.6 % (63 / 8)	100.0 % (59 / 1)	97.0% (90 / 7)	89.2% (311 / 652)	100.0 % (15 / 17)
Results from VN	Results from VNN-COMP'24 Brix et al. (2024)							
o Q CDOWN	100.0%	100.0%	87.8 %	98.6%	100.0%	98.0%	100.0%	100.0%
α - β -CROWN	(139 / 47)	(30/32)	(24 / 134)	(63 / 8)	(59/1)	(91/7)	(421 / 659)	(15 / 17)
NeuralSAT	98.9%	100.0%	87.2%	98.6%	100.0%	98.0%	90.5%	100.0%
NeuraisAi	(138 / 46)	(30/32)	(23 / 134)	(63 / 8)	(59/1)	(91/7)	(327 / 650)	(15 / 17)
D.D AT	98.9%	93.5%	83.3%	98.6%	100.0%	97.0%	79.9%	100.0%
PyRAT	(137/47)	(30/28)	(22 / 128)	(63 / 8)	(59/1)	(91/6)	(277 / 586)	(15 / 17)
Marabou	96.2%	100.0%	86.7%	95.8%	100.0%	53.0%	62.5%	93.8%
	(134/45)	(30/32)	(22 / 134)	(62 / 7)	(59/1)	(46/7)	(300 / 375)	(13 / 17)
nnenum	99.5%	100.0%	14.4	_	98.3%	46.0%	89.2%	56.2%
	(139 / 46)	(30/32)	(20/6)	(- / -)	(59 / 0)	(44 / 2)	(321 / 642)	(2 / 16)

Table 2: Ablation studies.

(a) without (X) vs. with $(\ensuremath{\checkmark})$ Input Refinement

(b)	GPU	vs.	CPU
[avg.	Time \downarrow (%S	olved ↑)]	

Innut Dofinament	aca	sxu	safenlp		
Input Refinement	X	✓ (Ours)	X	✓ (Ours)	
avg. #Sub- problems ↓ (max)	2 104.1 (133 438)	830.2 (36 134)	12 774.3 (293 303)	9 598.1 (180 015)	
%Solved ↑ (#Verified / #Falsified)	99.5 % (138 / 47)	99.5 % (138 / 47)	80.9 (299 / 575)	89.2% (311 / 652)	
avg. Time ↓	1.2 s	1.5s	5.1s	2.6 s	

Method (Batch Size)	acasxu	safenlp
CPU (1)	7.3s (95.2%)	3.6s (83.1%)
GPU (128) GPU (1024)	1.7s (99.5%) 1.5 (99.5%)	2.9s (86.8%) 2.6 s (89.2%)

GPU Acceleration and Batch Size We demonstrate the efficacy and speed up of the GPU acceleration by comparing the results of the acasxu and safenlp benchmark computed on a CPU (Tab. 2b): The GPU acceleration enables a significant speed up, which allows the verification of more instances within the allotted timeout, i.e., the GPU speeds up the verification by 79.4% on acasxu and 38.5% on safenlp.

Falsification Method We compare our set-based adversarial attack (Sec. 4.2) against the fast-gradient-sign method (FGSM) Goodfellow et al. (2015) on the safenlp benchmark. Tab. 3 shows the number of instances falsified within the first 50 iterations to compare the falsification strength without influence from the runtime. Our set-based adversarial attack outperforms FGSM by falsifying 60.3% more instances. Further, the main results (Tab. 1) show that in most benchmarks we match the falsification performance of the top tools.

Table 3: Falsification [#Falsified \uparrow (%Solved)].

Falsification	safenlp $(\leq 50 \ iter.)$
FGSM	257 (23.8%)
Ours	647 (59.9%)

6 Related Work

Neural Network Verification and Adversarial Attacks Most algorithms for neural network verification either (i) formulate an optimization or constraint satisfaction problem and apply an off-the-shelf solver, e.g., satisfiability modulo theories Katz et al. (2017); Duong et al. (2023); Wu et al. (2024) or (mixed-integer) linear programming Singh et al. (2019b); Müller et al. (2022), or (ii) use abstract interpretation or reachability analysis to enclose the intersection of the specification with the output of the neural network Gehr et al. (2018); Wang et al. (2018); Singh et al. (2019a); Wang et al. (2021); Kochdumper et al. (2023); Ladner & Althoff (2023); Lemesle et al. (2024).

The most common abstract domains or set representations are intervals Gehr et al. (2018), zonotopes Gehr et al. (2018), or polytopes Müller et al. (2022); Zhang et al. (2018); Singh et al. (2019a), that use linear relaxations of activation functions for the propagation. More complex set representations can enclose the output set more tightly, e.g., polynomial zonotopes Kochdumper et al. (2023), hybrid zonotopes Ortiz et al. (2023), star sets Bak (2021); however, due to higher computational cost, these approaches do not scale to large neural networks. The results of multiple abstract domains can be combined to obtain a tighter enclosure Lemesle et al. (2024). A popular abstract domain, DeepPoly/CROWN Singh et al. (2019a); Zhang et al. (2018), uses bounded polytopes, which are represented as the intersection of linear bounds and intervals. Further, gradient-based optimization can be applied to bounding parameters to tighten the computed enclosure Wang et al. (2021).

If formal verification is computationally infeasible, an alternative is to falsify neural networks using adversarial attacks. Often, gradient-based attacks are fast and effective at finding adversarial perturbations Goodfellow et al. (2015); Kurakin et al. (2017). Further, stronger adversarial examples can be computed with optimization-based approaches Carlini & Wagner (2017). Our set-based falsification does not require the gradient of the neural network while being faster to compute than an optimization problem.

Iterative Neural Network Refinement There are different iterative refinement approaches, e.g., refining bounds using (mixed-integer) linear programs Wang et al. (2018); Singh et al. (2019b); Yang et al. (2021), counterexample guided abstraction refinement Wu et al. (2024), and refining the hyperparameters of output set enclosure Xu et al. (2021); Ladner & Althoff (2023). Our novel specification-driven input refinement iteratively encloses the set of inputs that cause an intersection with an unsafe set in the output space. Therefore, our refinement does not require tuning hyperparameters or solving expensive (mixed-integer) linear programs. The refinement procedure uses dependencies between propagated zonotopes, which can be used to simplify computations for reachability analysis or identify falsifying states Kochdumper et al. (2020).

Further, our input refinement can be used to enclose the preimage of a neural network. Linear bounding techniques can be used to enclose preimages by solving multiple linear programs to compute bounds with respect to constraints or dimensions separately Kotha et al. (2023); Zhang et al. (2024). Conversely, we compute a zonotope enclosing the preimage that we optimize with an iterative procedure with respect to all constraints and dimensions simultaneously.

Branch-and-Bound Algorithms and Splitting Heuristics For large neural networks, most verification approaches are too conservative. Therefore, in practice, most verification approaches use a branch-and-bound procedure that recursively splits the verification problem into smaller subproblems that are easier to solve Brix et al. (2023; 2024), e.g., by exploiting the piece-wise linearity of the ReLU-activation function. There are various approaches and heuristics for splitting a verification problem, e.g., largest radius, largest approximation error and its effect on the output constraints Henriksen & Lomuscio (2021), gradient or sensitivity-based heuristics to estimate the impact of a neuron on the output Balunovic et al. (2019); Ladner & Althoff (2023), magnitude of the coefficient of linear relaxation Durand et al. (2022), largest unstable neuron in the first undecidable layer Yin et al. (2022), multi-neuron constraints Ferrari et al. (2022), least unstable neuron Duong et al. (2023).

7 CONCLUSION

In this paper, we construct a novel latent space for an iterative refinement procedure to speed up the formal verification of neural networks. The latent space is constructed from the propagation of projection-based set representations, e.g., zonotopes, through the layers of a neural network. Our procedure iteratively refines an enclosure of the set of unsafe inputs by using the latent space to constrain the input set with the unsafe set from the output space. We integrate our refinement procedure into a branch-and-bound neural network verification algorithm. In an extensive evaluation, we show a significant reduction in the number of recursive splits required for verification. Moreover, we show that our algorithm achieves competitive performance compared to the top-5 tools of the last neural network verification competition. In summary, our novel latent space presents a promising new direction for the formal verification of neural networks.

REFERENCES

- Matthias Althoff. *Reachability analysis and its application to the safety assessment of autonomous cars.* PhD thesis, Technische Universität München, 2010.
- Matthias Althoff. An introduction to CORA 2015. In *ARCH Workshop*, pp. 120–151, 2015.
- Matthias Althoff and Goran Frehse. Combining zonotopes and support functions for efficient reachability analysis of linear systems. In *CDC*, pp. 7439–7446, 2016.
 - Stanley Bak. nnenum: Verification of relu neural networks with optimized abstraction refinement. In *NASA formal methods symposium*, pp. 19–36, 2021.
 - Mislav Balunovic, Maximilian Baader, Gagandeep Singh, Timon Gehr, and Martin Vechev. Certifying geometric robustness of neural networks. In *NeurIPS*, volume 32, 2019.
 - Christopher Bishop. Pattern recognition and machine learning. Springer New York, NY, 2006.
 - Christopher Brix, Mark Niklas Müller, Stanley Bak, Taylor Johnson, and Changliu Liu. First three years of the international verification of neural networks competition (VNN-COMP). *STTT*, 25 (3):329–339, 2023.
 - Christopher Brix, Stanley Bak, Taylor Johnson, and Haoze Wu. The fifth international verification of neural networks competition (VNN-COMP 2024): Summary and results. *arXiv* preprint *arXiv*:2412.19985, 2024.
 - Rudy Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan Mudigonda. A unified view of piecewise linear neural network verification. In *NeurIPS*, volume 31, 2018.
 - Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE SSP*, pp. 39–57, 2017.
 - Pranav Singh Chib and Pravendra Singh. Recent advancements in end-to-end autonomous driving using deep learning: A survey. *IEEE T-IV*, 9(1):103–118, 2023.
 - Christophe Combastel. Zonotopes and Kalman observers: Gain optimality under distinct uncertainty paradigms and robust convergence. *Automatica*, 55:265–273, 2015.
 - Hai Duong, ThanhVu Nguyen, and Matthew Dwyer. A dpll (t) framework for verifying deep neural networks. *arXiv preprint arXiv:2307.10266*, 2023.
 - Serge Durand, Augustin Lemesle, Zakaria Chihani, Caterina Urban, and François Terrier. ReCIPH: Relational coefficients for input partitioning heuristic. In *WFVML*, 2022.
 - Claudio Ferrari, Mark Niklas Müller, Nikola Jovanović, and Martin Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. In *ICLR*, 2022.
 - Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *IEEE SSP*, pp. 3–18, 2018.
 - Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *HSCC*, pp. 291–305, 2005.
 - Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.
 - Patrick Henriksen and Alessio Lomuscio. DEEPSPLIT: An efficient splitting method for neural network verification via indirect effect analysis. In *IJCAI*, pp. 2549–2555, 2021.
- Ahmed Irfan, Kyle D. Julian, Haoze Wu, Clark Barrett, Mykel J. Kochenderfer, Baoluo Meng, and James Lopez. Towards verification of neural networks for small unmanned aircraft collision avoidance. In *DASC*, pp. 1–10, 2020.
 - Radoslav Ivanov, James Weimer, Rajeev Alur, George Pappas, and Insup Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *HSCC*, pp. 169–178, 2019.

- John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.
 - Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *CAV*, pp. 97–117, 2017.
 - Donald Knuth. *The Art of Computer Programming*, volume 1: Fundamental Algorithms. Addison-Wesley, 1997.
 - Niklas Kochdumper, Bastian Schürmann, and Matthias Althoff. Utilizing dependencies to obtain subsets of reachable sets. In *HSCC*, 2020.
 - Niklas Kochdumper, Christian Schilling, Matthias Althoff, and Stanley Bak. Open- and closed-loop neural network verification using polynomial zonotopes. In *NASA Formal Methods*, pp. 16–36, 2023.
 - Lukas Koller, Tobias Ladner, and Matthias Althoff. Set-based training for neural network verification. TMLR, 2025.
 - Suhas Kotha, Christopher Brix, Zico Kolter, Krishnamurthy Dvijotham, and Huan Zhang. Provably bounding neural network preimages. In *NeurIPS*, 2023.
 - Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *ICLR*, 2017.
 - Tobias Ladner and Matthias Althoff. Automatic abstraction refinement in neural network verification using sensitivity analysis. In *HSCC*, pp. 1–13, 2023.
 - Augustin Lemesle, Julien Lehmann, and Tristan Le Gall. Neural network verification with pyrat. *arXiv preprint arXiv:2410.23903*, 2024.
 - Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. PRIMA: general and precise neural network certification via scalable convex hull approximations. *PACMPL*, 6(POPL), 2022.
 - Joshua Ortiz, Alyssa Vellucci, Justin Koeln, and Justin Ruths. Hybrid zonotopes exactly represent relu neural networks. In *CDC*, pp. 5351–5357, 2023.
 - Alexander Schrijver. Theory of linear and integer programming. John Wiley & Sons, 1998.
 - Joseph Scott, Davide Raimondo, Giuseppe Roberto Marseglia, and Richard Braatz. Constrained zonotopes: A new tool for set-based estimation and fault detection. *Automatica*, 69:126–136, 2016.
 - Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *PACMPL*, 3(POPL), 2019a.
 - Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. Boosting robustness certification of neural networks. In *ICLR*, 2019b.
 - Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. In *NeurIPS*, pp. 6369–6379, 2018.
 - Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and Zico Kolter. Betacrown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *NeurIPS*, 34:29909–29921, 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 *CAV*, pp. 249–264, 2024.

- 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. In *ICLR*, 2021.
- Pengfei Yang, Renjue Li, Jianlin Li, Cheng-Chao Huang, Jingyi Wang, Jun Sun, Bai Xue, and Lijun Zhang. Improving neural network verification through spurious region guided refinement. In *TACAS*, pp. 389–408, 2021.
- Banghu Yin, Liqian Chen, Jiangchao Liu, and Ji Wang. Efficient complete verification of neural networks via layerwised splitting and refinement. *IEEE TCAD*, 41(11):3898–3909, 2022.
- Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *NeurIPS*, 31, 2018.
- Xiyue Zhang, Benjie Wang, and Marta Kwiatkowska. Provable preimage under-approximation for neural networks. In *TACAS*, 2024.
- Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *Proc. IEEE*, 111(3):257–276, 2023.

A APPENDIX – EVALUATION DETAILS

For the evaluation, we implement our verification algorithm in MATLAB using the CORA Althoff (2015) toolbox. To avoid an unfair comparison with the results of the VNN-COMP'24 Brix et al. (2024), we run all our evaluations on a laptop with inferior hardware compared to the VNN-COMP'24, i.e., Intel Core i7-13700H and a NVIDIA RTX 4070 Laptop GPU (8GB). The competition uses servers that are equipped with an NVIDIA A10G GPU with significantly more memory (24GB) (Brix et al., 2024, Sec. 2).

The selected benchmarks are taken from the VNN-COMP'24 Brix et al. (2024): acasxu a standard benchmark used by many prior works containing neural networks for airborne collision detection; collins-rul-cnn containing neural networks for condition-based maintenance; cora and metaroom containing neural networks for high-dimensional image classification; dist-shift containing neural networks for distribution shift detection; linearizenn containing neural networks for an autonomous aircraft taxiing system; safenlp containing neural networks for sentence classification; tllverifybench containing two-level lattice neural networks.

Our selection of the top-5 tools is based on the ranking after the tool updates that resolved many penalties due to incorrectly formatted counterexamples (Brix et al., 2024, Tab. 35).

B APPENDIX – ENCLOSURE-GRADIENT SPLITTING HEURISTIC

In each iteration of our verification algorithm (Alg. 1), we split each input set along the middle of an input dimension or exploit the piece-wise linearity of an unstable ReLU-neuron. Typical gradient-based splitting heuristics use the local gradient of the neural network to rank the importance of different input dimensions or neurons Balunovic et al. (2019); Ladner & Althoff (2023). The local gradient does not explicitly consider effects of splitting input dimensions or neurons on the conservatism of the enclosure, i.e., a split does not guarantee a tighter enclosure. Therefore, we compute the gradient of the size of the output set with respect to the enclosure. Thus, with each split we can guarantee a reduction in the conservatism of the enclosure. We use the F-radius to measure the size of the output enclosure $\mathcal{Y} = \langle c_y, G_y \rangle_Z = \text{enclose}(\Phi, \mathcal{X})$, i.e., the F-radius is defined as the

Frobenius norm of the generator matrix (Combastel, 2015, Def. 3): $\|\mathcal{Y}\|_F := \sqrt{\sum_{i=1}^n \sum_{j=1}^q G_{y(i,j)}^2}$.

Intuitively, we use the gradient of the F-radius to measure the contribution of an input dimension or an approximation error to the overall size of the output enclosure. For an input dimension $i \in [n_0]$, we compute the score

$$s(i) = r_{(i)} \nabla_{r_{(i)}} \|\mathcal{Y}\|_F, \tag{10}$$

Table 4: Splitting Heuristics.

Heuristic + Input Refinement Prop. 2	%Solved ↑ (#Verified / #Falsified)	avg. Time ↓	avg. #Subproblems (max)↓
Local Gradient	acasxu 98.4% (136 / 47)	3.7s	11 546.18 (430 403)
Enclosure Gradient (Ours)	99.5% (138 / 47)	1.5s	830.2 (36 134)
	safenlp		
Local Gradient	87.4% (297 / 647)	2.8s	6 334.1 (177 123)
Enclosure Gradient (Ours)	89.2 % (311 / 652)	2.6s	4700.3 ¹ (138 367)

where the $r = |G_x| \mathbf{1}$ is the radius of the input set $\mathcal{X} = \langle c_x, G_x \rangle_Z \subset \mathbb{R}^{n_0}$. If the k-th layer is a ReLU-activation layer, for $k \in [\kappa]$, we compute the score for the i-th neuron, for $i \in [n_k]$, by

$$s(L_k, i) = e_{k(i)} \nabla_{e_{k(i)}} \|\mathcal{Y}\|_F,$$
 (11)

where $e_k = 1/2 (\overline{e}_k - \underline{e}_k)$ is the radius of the approximation error. In each iteration, we split the input dimension or the neuron with the largest score.

In Tab. 4, we compare our enclosure-gradient heuristic against a local-gradient heuristic defined as Ladner & Althoff (2023)

$$s(i) = r_{(i)} \|\nabla_{r_{(i)}} y\|_2 \qquad \qquad s(L_k, i) = e_{k(i)} \|\nabla_{e_{k(i)}} y\|_2, \tag{12}$$

where $y \in \mathbb{R}^{n_{\kappa}}$ is the output of the neural network for the center of the input set, i.e., $y = \Phi(c_x)$. Our enclosure-gradient heuristic outperforms the local-gradient heuristic on the considered benchmarks (acasxu and safenlp), demonstrating its effectiveness.

C APPENDIX – IMPLEMENTATION TRICKS

We efficiently implement our algorithm with only matrix operations to take full advantage of GPU acceleration. Further, to make our algorithm practical, we reduce the memory footprint by only storing the bounds of the constrained zonotopes in the queue. We use zonotopes for the set propagation (line 1). However, we compute the bounds of a constrained zonotope before an enqueue operation.

Computing the bounds of constrained zonotopes requires solving two linear programs for each dimension, which limits GPU acceleration. Hence, we avoid solving linear programs by efficiently approximating the bounds of a constrained zonotope by approximating the bounds of the constrained hypercube (Prop. 3). The approximation is inspired by the Fourier-Motzkin elimination (Schrijver, 1998, Sec. 12.2).

To avoid clutter, we introduce further notation: For a matrix $A \in \mathbb{R}^{n \times m}$, we denote the matrix with all non-positive entries set to zero with $A^+ \in \mathbb{R}_{\geq 0}$, i.e., $A^+ = 1/2$ (A + |A|); the matrix $A^- \in \mathbb{R}_{\geq 0}$ is defined analogously.

Proposition 3 (Bounds of Bounded Polytope). A constrained hypercube $\mathcal{P} = \langle C, d \rangle_H \cap \left[\underline{\beta}_0, \overline{\beta}_0\right] \subset \mathbb{R}^q$ with $C \in \mathbb{R}^{p \times q}$ and $d \in \mathbb{R}^p$ is enclosed by $[\underline{\beta}, \overline{\beta}]$, i.e., $\mathcal{P} \subseteq [\underline{\beta}, \overline{\beta}]$. For each dimension $j \in [q]$, let $\Sigma_{\backslash \{j\}} := I_q - \mathbf{e}_j \mathbf{e}_j^{\mathsf{T}}$, and for each $i \in [p]$, let $\underline{C}_{(i,j)} := I_q - \mathbf{e}_j \mathbf{e}_j^{\mathsf{T}}$.

¹To provide a fair comparison, we only compute the number of verified subproblems across the instances solved by both heuristics. The additional instances solved with the enclosure-gradient heuristic would inflate the number of verified subproblems, i.e., avg. #Subproblems: 9 598.2 and max. #Subproblems of 180 015.

$$(d_{(i)} - C_{(i,\cdot)}^+ \sum_{\{j\}} \underline{\beta}_0 - C_{(i,\cdot)}^- \sum_{\{j\}} \overline{\beta}_0) / C_{(i,j)}. \text{ The bounds of } \mathcal{P} \text{ are computed by}$$

$$\underline{\beta}_{(j)} = \max \{\underline{C}_{(i,j)} \mid i \in [p] \colon C_{(i,j)} < 0\} \cup \{\underline{\beta}_{0(j)}\},$$

$$\overline{\beta}_{(j)} = \min \{\underline{C}_{(i,j)} \mid i \in [p] \colon C_{(i,j)} > 0\} \cup \{\overline{\beta}_{0(j)}\}.$$

Proof. We fix a point $\beta \in \mathcal{P}$ and indices $i \in [p]$ and $j \in [q]$ and show that $\beta \in [\underline{\beta}, \overline{\beta}]$. From the definition of a polytope, it holds that: $C_{(i,\cdot)} \beta \leq d_{(i)}$. By rearranging the terms, we obtain

$$C_{(i,\cdot)} \beta \leq d_{(i)} \quad \iff \quad C_{(i,j)} \beta_{(j)} \leq d_{(i)} - \sum_{k \in [q] \setminus \{j\}} C_{(i,k)} \beta_{(k)} = d_{(i)} - C_{(i,\cdot)} \sum_{k \in [j] \setminus \{j\}} \beta.$$

We split cases on the sign of $C_{(i,j)}$:

Case 1 ($C_{(i,j)} < 0$). We obtain a lower bound for $\beta_{(j)}$:

$$\frac{d_{(i)} - C_{(i,\cdot)} \sum_{i \in J} \beta}{C_{(i,j)}} \le \beta_{(j)}.$$

Using the initial bounds, i.e., $\underline{\beta}_{0(i)} \leq \beta_{(j)} \leq \overline{\beta}_{0(j)}$, we can approximate the bounds of $\beta_{(j)}$:

$$\begin{split} \beta_{(j)} &\overset{(C_{(i,j)}<0)}{\geq} \frac{d_{(i)} - C_{(i,\cdot)} \sum_{\backslash \{j\}} \beta}{C_{(i,j)}} \geq \min_{\tilde{\beta} \in \mathcal{P}} \frac{d_{(i)} - C_{(i,\cdot)} \sum_{\backslash \{j\}} \tilde{\beta}}{C_{(i,j)}} \geq \min_{\tilde{\beta} \in \left[\underline{\beta}_0, \overline{\beta}_0\right]} \frac{d_{(i)} - C_{(i,\cdot)} \sum_{\backslash \{j\}} \tilde{\beta}}{C_{(i,j)}} \\ &= \frac{d_{(i)} - \min_{\tilde{\beta} \in \left[\underline{\beta}_0, \overline{\beta}_0\right]} C_{(i,\cdot)} \sum_{\backslash \{j\}} \tilde{\beta}}{C_{(i,j)}} = \frac{d_{(i)} - C_{(i,\cdot)}^+ \sum_{\backslash \{j\}} \underline{\beta}_0 - C_{(i,\cdot)}^- \sum_{\backslash \{j\}} \overline{\beta}_0}{C_{(i,j)}} = \underline{C}_{(i,j)}. \end{split}$$

Case 2 ($C_{(i,j)} > 0$). We obtain an upper bound for $\beta_{(j)}$:

$$\beta_{(j)} \le \frac{d_{(i)} - C_{(i,\cdot)} \sum_{\{j\}} \beta}{C_{(i,j)}}.$$

Using the initial bounds, i.e., $\underline{\beta}_{0(j)} \leq \beta_{(j)} \leq \overline{\beta}_{0(j)}$, we can approximate the bounds of $\beta_{(j)}$:

$$\beta_{(j)} \overset{(C_{(i,j)}>0)}{\leq} \frac{d_{(i)} - C_{(i,\cdot)} \sum_{\backslash \{j\}} \beta}{C_{(i,j)}} \leq \max_{\tilde{\beta} \in \mathcal{P}} \frac{d_{(i)} - C_{(i,\cdot)} \sum_{\backslash \{j\}} \tilde{\beta}}{C_{(i,j)}} \leq \max_{\tilde{\beta} \in \left[\underline{\beta}_0, \overline{\beta}_0\right]} \frac{d_{(i)} - C_{(i,\cdot)} \sum_{\backslash \{j\}} \tilde{\beta}}{C_{(i,j)}}$$

$$= \frac{d_{(i)} - \min_{\tilde{\beta} \in \left[\underline{\beta}_0, \overline{\beta}_0\right]} C_{(i,\cdot)} \sum_{\backslash \{j\}} \tilde{\beta}}{C_{(i,j)}} = \frac{d_{(i)} - C_{(i,\cdot)}^+ \sum_{\backslash \{j\}} \underline{\beta}_0 - C_{(i,\cdot)}^- \sum_{\backslash \{j\}} \overline{\beta}_0}{C_{(i,j)}} = \underline{C}_{(i,j)}.$$

Each row i of the constraint matrix C generates an inequality for each dimension j of β . Thus, with $\beta \in \left[\underline{\beta}_0, \overline{\beta}_0\right]$, we set $\underline{\beta}_{(j)}$ to maximum lower bound and $\overline{\beta}_{(j)}$ to minimum lower bound.

Note that Prop. 3 can be iterated: For $\mathcal{P}=\langle C,d\rangle_H\cap\left[\underline{\beta}_0,\overline{\beta}_0\right]$ with bounds $\left[\underline{\beta},\overline{\beta}\right]$, it holds that $\mathcal{P}=\langle C,d\rangle_H\cap\left[\underline{\beta},\overline{\beta}\right]$; thus, Prop. 3 can be applied again to obtain potentially smaller bounds. Fig. 4 illustrates the approximated bounds and exact bounds of a bounded polytope.

The bounds of a constrained zonotope $\mathcal{Z}|_{C \leq d} \subset \mathbb{R}^n$ with $\mathcal{Z} = \langle c, G \rangle_Z$ can be approximated by applying an affine map to the bounds $[\beta, \overline{\beta}] \subseteq \overline{\mathcal{B}_q}$ of its constrained hypercube:

$$\mathcal{Z}|_{C \le d} \subseteq c \oplus G^{-}\left[\underline{\beta}, \overline{\beta}\right] \oplus G^{+}\left[\underline{\beta}, \overline{\beta}\right]. \tag{13}$$

In Tab. 5, we evaluate the effects of our fast bounding a constraint zonotope (Prop. 3) on the verification accuracy. We turned off the timeout because with the exact bounds, we could not verify any instance within the allotted time. Both approaches can verify all instances, and as expected, computing the exact bounds is significantly slower, i.e., the average verification time is 823.3s vs. 1.0s for our fast bound approximation. Further, the exact bounds only marginally reduce the number of verified subproblems, i.e., on average 276.98 vs. 277.58 for the fast bound approximation, which indicates that our approximated bounds are sufficiently tight for the verification of neural networks.

²Computing the exact bounds of a high-dimensional constrained zonotope is computationally expensive, thus, we can only compare our approximation on the first 45 instances of the acasxu benchmark (prop1).

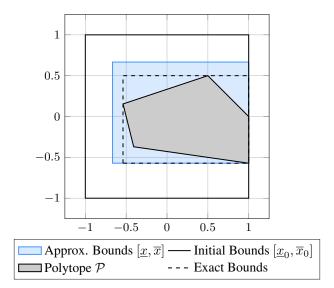


Figure 4: Illustrating the approximated bounds (Prop. 3) and the exact bounds of a bounded polytope.

Table 5: Approximation of Constraint Zonotope Bounds.

Exact Bounds	acasxu² ✓ X (Prop. 3)		
avg. #Sub- problems↓(max)	276.98 (1 045)	277.58 (1 051)	
$\%$ Solved \uparrow (#Verified / #Falsified) \uparrow	100% (45 / 0)	100% (45 / 0)	
avg. Time ↓	823.3s	1.0s	

D APPENDIX - PROOFS

Proposition 2. Given are a neural network $\Phi \colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_\kappa}$, an input set $\mathcal{X} = \langle c_x, G_x \rangle_Z \subset \mathbb{R}^{n_0}$ with $G_x \in \mathbb{R}^{n_0 \times q_0}$, and an unsafe set $\mathcal{U} = \langle A, b \rangle_H \subset \mathbb{R}^{n_\kappa}$. Let $\mathcal{Y} = \langle c_y, G_y \rangle_Z = \mathsf{enclose}(\Phi, \mathcal{X})$ be an enclosure of the output set with $G_y \in \mathbb{R}^{n_\kappa \times q_\kappa}$. We enclose all unsafe inputs with

$$\{x \in \mathcal{X} \mid \Phi(x) \in \mathcal{U}\} \subseteq \mathcal{X}|_{C < d}$$

where
$$C := A G_{y(\cdot,[q_0])}$$
 and $d := b - A c_y + \left| A G_{y(\cdot,[q_\kappa]\setminus[q_0])} \right| \mathbf{1}$.

Proof. We fix an unsafe input $x \in \{x \in \mathcal{X} \mid \Phi(x) \in \mathcal{U}\}$. Let $y = \Phi(x)$ be its corresponding output. We use the definition of the unsafe set \mathcal{U} and the definition of a zonotope, i.e., there are factors $\beta \in \mathcal{B}_{q_{\kappa}}$ s.t.

$$y = c_v + G_v \beta. (14)$$

For an unsafe output, we have the following inequality:

$$y \in \mathcal{U} \iff A(c_y + G_y \beta) \le b \iff AG_y \beta \le b - Ac_y.$$

Please note that the first q_0 factors of the output y are the factors of the input x, i.e., $x = c_x + G_x \beta_{([q_0])}$. Our goal is to constrain the factors $\beta_{([q_0])}$. Therefore, we rearrange the terms

$$A G_{y} \beta \leq b - A c_{y} \iff A G_{y(\cdot,[q_{0}])} \beta_{([q_{0}])} + A G_{y(\cdot,[q_{\kappa}] \setminus [q_{0}])} \beta_{([q_{\kappa}] \setminus [q_{0}])} \leq b - A c_{y}$$

$$\iff C \beta_{([q_{0}])} \leq b - (A c_{y} + A G_{y(\cdot,[q_{\kappa}] \setminus [q_{0}])} \beta_{([q_{\kappa}] \setminus [q_{0}])}).$$

We bound the right-hand side to get to obtain constraints $C \beta_{(q_0)} \leq d$ on the factors of x:

$$C \,\beta_{([q_0])} \le b - (A \, c_y + A \, G_{y(\cdot, [q_\kappa] \setminus [q_0])} \,\beta_{([q_\kappa] \setminus [q_0])}) \le b - A \, c_y + \left| A \, G_{y(\cdot, [q_\kappa] \setminus [q_0])} \right| \mathbf{1} = d.$$

Thus, we conclude that the input x is contained in the constrained input set \mathcal{X} :

$$x \in \mathcal{X}|_{C \leq d}$$
.

The following corollary proves the soundness of our refinement for neural network verification and falsification.

Corollary 1. Given a neural network $\Phi \colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_\kappa}$, an input set $\mathcal{X} \subset \mathbb{R}^{n_0}$, and unsafe set $\mathcal{U} = \langle A, b \rangle_H \subset \mathbb{R}^{n_\kappa}$, the refinement of the input set (Prop. 2) can be used for verification and falsification

$$\Phi(\mathcal{X}|_{C \leq d}) \cap \mathcal{U} = \emptyset \iff \Phi(\mathcal{X}) \cap \mathcal{U} = \emptyset,$$
$$\exists x \in \mathcal{X}|_{C \leq d} \colon \Phi(x) \in \mathcal{U} \iff \exists x \in \mathcal{X} \colon \Phi(x) \in \mathcal{U}.$$

Proof. With Prop. 2 we obtain $\mathcal{X} \setminus \mathcal{X}|_{C \leq d} \subseteq \{x \in \mathcal{X} \mid \Phi(x) \notin \mathcal{U}\}$, and therefore, we have $\Phi(\mathcal{X} \setminus \mathcal{X}|_{C \leq d}) \cap \mathcal{U} = \emptyset$ and $\neg(\exists x \in \mathcal{X} \setminus \mathcal{X}|_{C \leq d} \colon \Phi(x) \in \mathcal{U})$. Thus,

$$\Phi(\mathcal{X}|_{C \leq d}) \cap \mathcal{U} = \emptyset \iff \Phi(\mathcal{X}|_{C \leq d}) \cap \mathcal{U} = \emptyset \land \Phi(\mathcal{X} \setminus \mathcal{X}|_{C \leq d}) \cap \mathcal{U} = \emptyset
\iff (\Phi(\mathcal{X}|_{C \leq d}) \cup \Phi(\mathcal{X} \setminus \mathcal{X}|_{C \leq d})) \cap \mathcal{U} = \emptyset
\iff \Phi(\mathcal{X}) \cap \mathcal{U} = \emptyset,$$

$$\exists x \in \mathcal{X}|_{C \leq d} \colon \Phi(x) \in \mathcal{U} \iff \exists x \in \mathcal{X}|_{C \leq d} \colon \Phi(x) \in \mathcal{U} \lor \exists x \in \mathcal{X} \setminus \mathcal{X}|_{C \leq d} \colon \Phi(x) \in \mathcal{U}$$
$$\iff \exists x \in (\mathcal{X}|_{C \leq d} \cup \mathcal{X} \setminus \mathcal{X}|_{C \leq d}) \colon \Phi(x) \in \mathcal{U}$$
$$\iff \exists x \in \mathcal{X} \colon \Phi(x) \in \mathcal{U}.$$

E DISCLOSURE – USAGE OF LARGE LANGUAGE MODELS (LLMS)

We employed a large language model (LLM) as a general-purpose tool to refine the writing and enhance clarity of expression. All research ideas, methodology, analysis, and conclusions are solely the work of the authors.