TRAINING MIXED-MONOTONE NEURAL NETWORKS FOR VERIFICATION WITHOUT SET PROPAGATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Most formal verification approaches for neural networks conservatively enclose the output set of the neural network. However, computing a tight output enclosure is in general computationally hard. We sidestep this challenge with a novel neural network architecture that realizes mixed-monotone functions as the difference of monotone subnetworks. By utilizing the theory on mixed-monotone systems, this reduces the computation of a tight output enclosure from complicated set propagation to two simple forward passes of opposite corners of an input interval. Moreover, because the size of resulting enclosure is a closed-form function of the weights of the neural network, we can directly minimize it during training. We prove a lower bound on the size of any such enclosure, given by the monotone variation of the represented function, which makes the tightness of the enclosure measurable against an optimum. Our evaluation uses regression and dynamical-system benchmarks, to demonstrate that our training reaches within $0.5\,\%$ of this lower bound at no cost in prediction accuracy. Computing our enclosure is orders of magnitude faster than CROWN and α-CROWN and on deep networks it is also tighter than zonotope propagation and CROWN, and stays within $5\,\%$ of α-CROWN.

1 INTRODUCTION

Formal verification of neural networks can decide whether the output of a neural network stays within a specification for all inputs in a given set. This is a prerequisite wherever learned components enter safety-critical control loops, for example in autonomous driving (Ye et al., 2022). The exact image of a neural network is in general non-convex, and complete verifiers based on SMT or mixed-integer programming scale poorly in the number of neurons (Katz et al., 2017). Practical tools therefore enclose that image by propagating a symbolic set layer by layer, e.g., using intervals (Gowal et al., 2019), zonotopes (Girard, 2005; Singh et al., 2018; Koller et al., 2026), linear relaxations (Zhang et al., 2018) or polynomial zonotopes (Kochdumper & Althoff, 2021; Kochdumper et al., 2023), and pay an abstraction error at every unstable activation. That abstraction error accumulates with depth of the neural network and with the size of the input set. The abstraction error can be reduced with additional computations, e.g., higher-order abstractions (Kochdumper et al., 2023; Ladner & Althoff, 2023) or branch-and-bound (Bunel et al., 2020), creating a tradeoff between small abstraction errors and high computational cost.

Main idea We overcome this tradeoff by constructing a neural network to be a mixed-monotone function, i.e., the difference of two monotone subnetworks, which is visualized in Fig. 1. This allows us to use the theory on mixed-monotone systems to enclose the image of the neural network for an input interval without any set propagation. We only require two forward passes of opposite corners of the input interval. This is independent of the depth and width of the neural network and the number of input dimensions.

Moreover, we use the training of the neural network to explicitly optimize the tightness of the computed enclosure. Our results show that the neural network training drives the learned decomposition onto its theoretical optimum, to within $0.5\,\%$ of a theoretical lower bound at no cost in accuracy. The computation of our enclosure is orders of magnitude faster than CROWN and α-CROWN and on deep neural networks it is also tighter than zonotope propagation and CROWN, and stays within $5\,\%$ of α-CROWN.

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.

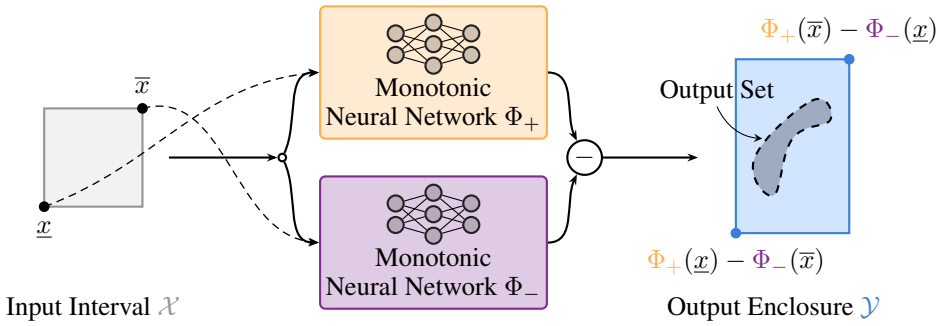


Figure 1: Our novel neural network architecture, which realizes the difference of two monotone neural networks to admit the fast and efficient computation of an output enclosure without requiring a set propagation.

Related work Monotonicity of neural networks can be achieved using sign-constrained weights (Sill, 1997; Daniels & Velikova, 2010), lattice networks (You et al., 2017), or certification of monotonicity after training (Liu et al., 2020; Sivaraman et al., 2020); we build on the sign-constrained neural networks of Runje & Shankaranarayana (2023) and the Lipschitz-residual neural network of Kitouni et al. (2023). That literature is motivated by fairness, interpretability and inductive bias; exploiting the enforced monotonicity has not been developed into a verification method.

Mixed-monotone approaches combine a decomposition of the plant with a separately computed inclusion function for the network (Jafarpour et al., 2023), or decompose a given neural network (Meyer, 2022); in both cases the decomposition is obtained after the neural network is fixed, and its tightness is inherited from whichever verifier supplies it. Classically, finding tight mixed-monotone decompositions is hard and the known constructions require structural assumptions such as a sign-stable Jacobian (Yang et al., 2019; Khajenejad & Yong, 2023). In this work, we train the neural network to be the decomposition function and we can directly optimize through neural network training the tightness of the decomposition, because the size of the enclosure is a closed-form function differentiable in the weights.

Moreover, through our view of the neural network as a decomposition function, we can make the tightness of the enclosure a training objective, without requiring set propagation during training. Therefore, a related field is certified training (Gowal et al., 2019; Xu et al., 2020; Müller et al., 2023), where abstraction errors are implicitly reduced by optimizing for certifiability or robustness using propagated sets during training. Further, set-based training (Koller et al., 2025; Wendl et al., 2026) is closer to our work, as it explicitly penalizes the size of the enclosure in the loss, but it still requires a differentiable set propagation in every training step, as does all certified training.

Contributions Our main contributions are:

- A neural network architecture for mixed-monotone functions, allowing the computation of an output enclosure using two simple forward passes.
- A theoretical lower bound on the size of the computed enclosures, which allows us to measure a learned decomposition against a functional optimum.
- A neural network training objective that directly minimizes the size of the enclosure.
- An evaluation on analytic and dynamical benchmarks, including open- and closed-loop experiments and comparisons with zonotope propagation, IBP, CROWN, and α-CROWN, showcasing the effectiveness of our approach.

2 Preliminaries

Notation We denote vectors with lowercase letters and matrices with uppercase letters. The i-th entry of a vector x is written as $x_{(i)}$ and the entry in the i-th row and the j-th column of a matrix $A \in \mathbb{R}^{n\times m}$ is written as $A_{(i,j)}$. For vectors $x, y \in \mathbb{R}^n$ of equal dimension, we write $x \leq y$

for the component-wise partial order, i.e., $x_{(i)} \leq y_{(i)}$ for all $i \in \{1, 2, \ldots, n\}$. 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 $|A|$ takes the elementwise absolute value of a matrix $A \in \mathbb{R}^{n \times m}$; we use $\text{sign}(\cdot)$ likewise. Sets are denoted by uppercase calligraphic letters. The set of all natural numbers up to $n \in \mathbb{N}$ is written as $[n] = \{1, 2, \ldots, n\}$. For a bounded set $\mathcal{S} \subset \mathbb{R}^m$, we write $\|\mathcal{S}\|_{\text{I}}$ for its interval norm (Althoff, 2023, Sec. 3.1), defined as the ℓ_1-norm of the diameter of its interval enclosure, i.e., $\|\mathcal{S}\|_{\text{I}} = \min_{[\underline{x}, \overline{x}] \supseteq \mathcal{S}} \mathbf{1}^\top (\overline{x} - \underline{x})$.

2.1 Monotone functions and mixed-monotone decompositions

In this work, we utilize the theory of mixed monotonicity to construct neural networks that admit easy-to-compute tight output enclosures.

A monotone function takes on its extremal values at two opposite corners of an input interval.

Definition 1 (Monotone function, (Coogan & Arcak, 2015, Def. 1)). *Given an interval $\mathcal{X}_0 \subseteq \mathbb{R}^n$, a function $f \colon \mathcal{X}_0 \to \mathbb{R}^m$ is*

(i) non-decreasing on $\mathcal{X}_0$ if $x \leq x' \implies f(x) \leq f(x')$, for all $x, x' \in \mathcal{X}_0$, and

(ii) non-increasing on $\mathcal{X}_0$ if $x \leq x' \implies f(x) \geq f(x')$, for all $x, x' \in \mathcal{X}_0$.

Mixed monotonicity generalizes monotonicity to functions that are recovered from monotone ones.

Proposition 1 (Enclosing a mixed-monotone function, (Coogan & Arcak, 2015, Def. 2)). *For $\mathcal{X}_0 \subseteq \mathbb{R}^n$, let $f \colon \mathcal{X}_0 \to \mathbb{R}^m$. A function $d_f \colon \mathcal{X}_0 \times \mathcal{X}_0 \to \mathbb{R}^m$ is a decomposition function for f iff for all $x_1, x_2 \in \mathcal{X}_0$*

(i) $d_f(x_1, x_1) = f(x_1)$, (ii) $d_f(\cdot, x_2)$ is non-decreasing, (iii) $d_f(x_1, \cdot)$ is non-increasing.

A function that admits a decomposition function is called mixed-monotone. The image of f over any interval $\mathcal{X} = [\underline{x}, \overline{x}] \subseteq \mathcal{X}_0$ can be enclosed by

$$f(\mathcal{X}) \subseteq [d_f(\underline{x}, \overline{x}), d_f(\overline{x}, \underline{x})].$$

Decomposition functions are not unique (Coogan, 2020); hence, finding a tight decomposition is a central challenge. In this work, we build neural networks to be mixed-monotone functions and we directly optimize the tightness of output enclosures through neural network training.

A standard decomposition function is the difference of two monotone functions (Coogan & Arcak, 2015, Ex. 1), i.e., $f(x) = g(x) - h(x)$, with monotone functions g and h, is a mixed-monotone function with decomposition function

$$d_f(x_1, x_2) = g(x_1) - h(x_2). \tag{1}$$

We build our neural network architecture based on this particular decomposition. Future work can explore more general forms of decomposition functions.

2.2 Monotonic Neural networks

Feed-forward networks $\Phi \colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_\kappa}$ are sequences of $\kappa \in \mathbb{N}$ layers. Each layer is either affine (linear layer) or an element-wise nonlinear activation function (nonlinear layer). In this work, we consider sign-constrained neural networks that can only represent monotonic functions.

Definition 2 (Sign-Constrained Neural Network, (Runje & Shankaranarayana, 2023, Def. 1)). *For an input $x \in \mathbb{R}^{n_0}$, the output of a sign-constrained neural network $y = \Phi(x) \in \mathbb{R}^{n_\kappa}$ is*

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

where

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

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.

Any sign-constrained neural network represents a monotonic function (Runje & Shankaranarayana, 2023, Lem. 1). For the gradient computation during training the weights remain unconstrained (Appendix A.11). The activation functions can be used determine whether the function represented by a monotonic neural network is convex or concave (Runje & Shankaranarayana, 2023, Cor. 3).

The goal of neural network verification is, 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 output set $\mathcal{U} \subset \mathbb{R}^{n_\kappa}$ to prove that all inputs are safe, i.e., $\forall x \in \mathcal{X}\colon \Phi(x) \notin \mathcal{U}$. To this end, we compute an enclosure $\mathcal{Y}$ of the output of the neural network (i.e., $\mathcal{Y} \supseteq \{\Phi(x) \mid x \in \mathcal{X}\}$) and try to show that the enclosure does not intersect with the unsafe set, i.e., $\mathcal{Y} \cap \mathcal{U} = \emptyset$.

2.3 Problem Statement

We want to build neural network architecture that admits an efficient computation of tight output enclosures to make neural network verification practical and scalable.

3 A monotone-difference architecture

We propose a neural network architecture that realizes mixed-monotone functions using the difference of two monotone neural subnetworks: Let Φ_+, $\Phi_-\colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_\kappa}$ be monotonic neural networks and we construct a neural network Φ as

$$\Phi = \Phi_+ - \Phi_-. \tag{2}$$

We use sign-constraints to realize the monotone neural subnetworks Φ_+, Φ_- (Def. 2). The neural network Φ is mixed-monotone by construction and we can use the decomposition function to enclose the image of the neural network.

Proposition 2. *Given an input set $\mathcal{X} = [\underline{x}, \overline{x}] \subseteq \mathbb{R}^{n_0}$, a neural network $\Phi = \Phi_+ - \Phi_-$, which is the difference two monotonic neural networks Φ_+, Φ_-, we can enclose the output set of the neural network by*

$$\texttt{enclose}(\Phi, \mathcal{X}) = [d_\Phi(\underline{x}, \overline{x}), d_\Phi(\overline{x}, \underline{x})] \supseteq \Phi(\mathcal{X}),$$

where

$$d_\Phi(x_1, x_2) = \Phi_+(x_1) - \Phi_-(x_2).$$

Proof. The neural network Φ is mixed monotone (Coogan & Arcak, 2015, Ex. 1), hence we can enclose the output using the decomposition function (Prop. 1). □

The computation of the enclosure only requires two forward passes through the neural network via the decomposition function, instead of requiring a complicated set propagation.

Computational Complexity Prop. 2 uses two forward passes to compute an enclosure; hence, the computation takes time $\mathcal{O}(n_{\max}^2\,\kappa)$, where $n_{\max} = \max_{k\in[\kappa]} n_k$ denotes the largest layer of the neural network, which is as expensive as a single point evaluation up to a constant factor. In contrast, propagating a zonotope through the same neural network takes $\mathcal{O}(n_{\max}^2\,q\,\kappa)$ time, where the number of generators grows with the size of the neural network, i.e., $q \in \mathcal{O}(n_0 + \sum_{k\in[\kappa]} n_k)$ (Koller et al., 2025, Sec. 4.4); linear relaxations, e.g., CROWN (Zhang et al., 2018) or DeepPoly (Singh et al., 2019), have a similar time complexity (Koller et al., 2025, Sec. 4.4). Our enclosure saves the entire factor q, that is linear in the number of neurons of the neural network.

4 Verification-aware training

The decomposition of a mixed-monotone function is not unique, and different decompositions of the same function yield enclosures of different size. Fig. 2 compares two decompositions, which yield enclosures of different sizes. First, we use an example to demonstrate the non-uniqueness of a decomposition and its effects on the size of the enclosure, before we explain, how we use neural network training to optimize the tightness of an enclosure.

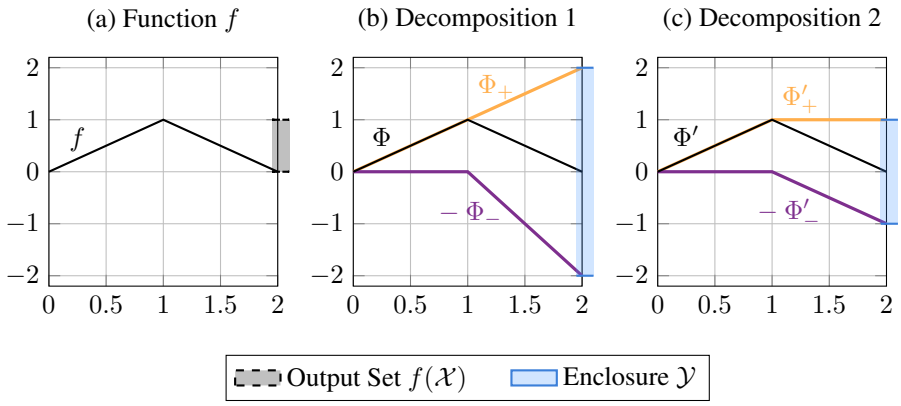


Figure 2: The size of an enclosure depends on the decomposition: For the same function f, two decompositions yield enclosures of different sizes.

Example 1 (Comparing Decompositions)**.** *Let $\mathcal{X} = [0, 2]$ be the input interval and let $f(x) = 1 - |x - 1|$ be a function (see Fig. 2(a)). The exact image of f (which generally is not computable) has size 1, i.e., $\|f(\mathcal{X})\|_{\mathrm{I}} = \|[0, 1]\|_{\mathrm{I}} = 1$.*

The function f can be represented by the mixed-monotone neural network $\Phi = \Phi_+ - \Phi_-$, with monotone subnetworks (see Fig. 2(b))

$$\Phi_+(x) = x, \qquad \Phi_-(x) = 2\,\max(x - 1, 0). \tag{3}$$

Using Prop. 1, we can enclose the output set of Φ by

$$\texttt{enclose}(\Phi, \mathcal{X}) = [\Phi_+(0) - \Phi_-(2), \Phi_+(2) - \Phi_-(0)] = [-2, 2].$$

The size of the enclosure is $\|\texttt{enclose}(\Phi, \mathcal{X})\|_{\mathrm{I}} = 4$.

In contrast, another mixed-monotone neural network $\Phi' = \Phi'_+ - \Phi'_-$ can also represent the function f with monotone subnetworks (see Fig. 2(c))

$$\Phi'_+(x) = \min(x, 1), \qquad \Phi'_-(x) = \max(x - 1, 0). \tag{4}$$

The computed enclosure is smaller, i.e.,

$$\texttt{enclose}(\Phi', \mathcal{X}) = \left[\Phi'_+(0) - \Phi'_-(2), \Phi'_+(2) - \Phi'_-(0)\right] = [-1, 1],$$

which has size $\|\texttt{enclose}(\Phi', \mathcal{X})\|_{\mathrm{I}} = 2$.

Training Loss Our goal is to directly optimize the tightness of the enclosure during training. To this end, we combine a standard regression loss (i.e., mean-squared error) with a penalty of the enclosure size for a fixed input perturbation radius $\epsilon \in \mathbb{R}^{n_0}$. For an input $x_j \in \mathbb{R}^{n_0}$ with target $t_j \in \mathbb{R}^{n_\kappa}$, we compute the training loss

$$L = \underbrace{\|\Phi(x_j) - t_j\|_2^2}_{\text{Regression Loss}} + \beta \underbrace{\|\texttt{enclose}(\Phi, [x_j - \epsilon, x_j + \epsilon])\|_{\mathrm{I}}}_{\text{Enclosure Size}}. \tag{5}$$

In principle, the training loss is similar to set-based training (Koller et al., 2025; Wendl et al., 2026): The training objective penalizes the size of an output enclosure. However, we do not require set propagations during training, which makes the training significantly more efficient.

Evaluating the training loss (5) only requires three forward passes, i.e., one for the regression loss and two for the enclosure (Prop. 2); hence, a training iteration takes $\mathcal{O}(n_{\max}^2\,\kappa)$ time, which is as expensive as standard point-based training up to a constant factor. In contrast, a set-based training iteration takes $\mathcal{O}(n_{\max}^2\,q\,\kappa)$ time (Koller et al., 2025, Sec. 4.4), i.e., our training saves the same factor q as our enclosure computation.

In the following, we derive a lower bound of the enclosure size, which motivates the training objective and serves as a reference to measure the tightness of computed enclosures.

Enclosure Size For a scalar function, the variation over an interval measures its total rise and fall (Cohn, 1993, Sec. 4.4); we use a multivariate extension along monotone sequences (Clarkson & Adams, 1933). For that, we define the collection of all finite sequences, i.e., for an interval $\mathcal{X} = [\underline{x}, \overline{x}] \subseteq \mathbb{R}^{n_0}$, let $\mathcal{S} \coloneqq \left\{\{x_j\}_{j=0}^{k} \mid k \in \mathbb{N}, \underline{x} \leq x_0 \leq x_1 \leq \cdots \leq x_k \leq \overline{x}\right\}$. The finite variation of a neural network Φ for each output dimension $i \in [n_\kappa]$ is defined as (Cohn, 1993, Sec. 4.4)

$$\nu_\Phi(\mathcal{X})_{(i)} = \sup \left\{ \sum_j \left| \Phi(x_j)_{(i)} - \Phi(x_{j-1})_{(i)} \right| \,\middle|\, \{x_j\} \in \mathcal{S} \right\}. \tag{6}$$

The variation sums the change of the neural network along the worst monotone sequence, which has to be covered by the monotonic subnetworks of the decomposition; hence, no decomposition can yield an enclosure smaller than the variation.

Proposition 3 (Smallest enclosure)**.** *Given an input interval $\mathcal{X} = [\underline{x}, \overline{x}] \subseteq \mathbb{R}^{n_0}$, a neural network $\Phi = \Phi_+ - \Phi_-$, where Φ_+, Φ_- are monotonic on $\mathcal{X}$, then*

$$\mathbf{1}^\top \nu_\Phi(\mathcal{X}) \leq \|\texttt{enclose}(\Phi, \mathcal{X})\|_{\mathrm{I}}.$$

Moreover, for $\mathcal{X}$ there exists $\Phi' = \Phi'_+ - \Phi'_-$ with monotonic Φ'_+, Φ'_- such that equality holds.

Proof. See Appendix A.1. □

The lower bound only depends on the function represented by the neural network and every decomposition has to cover the variation of this represented function.

The monotone subnetworks are universal approximators of continuous non-decreasing functions (Runje & Shankaranarayana, 2023, Thm. 7), hence, our architecture can realize enclosures arbitrarily close to the lower bound on a fixed input interval.

5 Evaluation

We implemented our mixed-monotone neural network architecture in Matlab using the CORA toolbox (Althoff, 2015).

Our evaluation is structured as follows: We first assess prediction accuracy of mixed-monotone neural networks, then compare enclosure tightness and computational cost, before showcasing the effectiveness of verification-aware training. Finally, we demonstrate the superior scalability of our approach across the number of input dimensions, number of neural network layers and evaluate the enclosure tightness in a closed-loop setting.

Benchmarks We evaluate our approach on an analytic regression task, two dynamical systems (Pendulum and Tanks), and a synthetic dimension sweep. The analytic task has a closed-form monotone variation and thus an exact reference for the lower bound of Prop. 3; the two dynamical systems cover a cooperative (Tanks) and a non-cooperative (Pendulum) plant for the closed-loop evaluation; and the dimension sweep isolates the effect of the input dimension by keeping the target's intrinsic complexity fixed. The benchmark definitions and experimental details are provided in Appendix A.2 to A.4.

Metrics We report prediction accuracy using R^2-scores. To measure the tightness of a computed enclosure, we report the ratio of its size to the size of the exact image, i.e., for an input set $\mathcal{X}$ and enclosure $\mathcal{Y} \supseteq \Phi(\mathcal{X})$, we report $\|\mathcal{Y}\|_{\mathrm{I}} / \|\Phi(\mathcal{X})\|_{\mathrm{I}}$. Since computing the exact image is generally intractable, we underapproximate its size using samples. Since $\|\Phi(\mathcal{X})\|_{\mathrm{I}} \leq \mathbf{1}^\top \nu_\Phi(\mathcal{X})$ by Prop. 3, this tightness also bounds the gap to the lower bound.

5.1 Prediction accuracy

Our sign-constrained architecture maintains high prediction accuracy across the tested benchmarks and number of neural network layers, showing that the monotonicity constraints do not prevent effective training. Tab. 1 compares it with an unconstrained baseline under the same training protocol, with differences in mean R^2 of at most 0.011 across the reported configurations. Additional analytic-task results, including the Lipschitz-constrained architectures, are provided in Tab. 6.

Table 1: Comparing prediction accuracy across different benchmarks. Mean ± standard deviation over ten seeds.

Benchmark	**#Layers κ**	**Accuracy ↑**	
		Baseline	Ours
Analytic	2	0.9837 ± 0.0018	$\mathbf{0.9858} \pm 0.0009$
	4	$\mathbf{0.9998} \pm 0.0001$	0.9895 ± 0.0051
	8	$\mathbf{0.9997} \pm 0.0003$	0.9964 ± 0.0035
Pendulum	3	$\mathbf{0.9942} \pm 0.0024$	0.9878 ± 0.0071
	8	$\mathbf{0.9931} \pm 0.0049$	0.9868 ± 0.0098
Tanks	3	$\mathbf{0.9997} \pm 0.0001$	0.9993 ± 0.0004
	8	$\mathbf{0.9985} \pm 0.0013$	0.9984 ± 0.0012

Table 2: Comparing enclosure tightness and runtimes on the analytic task. Tightness is mean ± standard deviation over five seeds. Runtimes are per input box, measured as the minimum over repetitions after warm-up. We evalute our approach within CORA as well as exported to `auto_LiRPA`.

#Layers κ	**Framework**	**Method**	**Tightness ↓**	**Runtime ↓**	
				*[*ms*]*	*[relative to ours]*
4	`auto_LiRPA`	Ours (exported)	1.1662 ± 0.1167	$\mathbf{0.12}$	$1\times$
		IBP	1.1662 ± 0.1167	1.21	$9.9\times$
		CROWN	1.1264 ± 0.0371	13.7	$112\times$
		α-CROWN	$\mathbf{1.0024} \pm 0.0025$	$1\,070$	$8\,750\times$
	CORA	Ours	1.1662 ± 0.1167	$\mathbf{0.01}$	$1\times$
		Zonotope	1.1592 ± 0.0483	0.17	$11.7\times$
8	`auto_LiRPA`	Ours (exported)	1.0534 ± 0.0805	$\mathbf{0.21}$	$1\times$
		IBP	1.0534 ± 0.0805	2.35	$11.2\times$
		CROWN	1.2640 ± 0.0608	53.8	$256\times$
		α-CROWN	$\mathbf{1.0022} \pm 0.0012$	$4\,330$	$20\,600\times$
	CORA	Ours	1.0534 ± 0.0805	$\mathbf{0.02}$	$1\times$
		Zonotope	1.4409 ± 0.0655	0.69	$32\times$

5.2 Verification tightness and runtimes

In Tab. 2, we compare the tightness and runtimes of our enclosure (Ours) with interval bound propagation (IBP; Gowal et al., 2019), zonotope propagation in CORA (Althoff, 2015), CROWN (Zhang et al., 2018), and α-CROWN (Xu et al., 2021) for neural networks trained on the analytic task; See Appendix A.13 and A.14 for details and additional results. Since the methods are implemented in different toolboxes, `auto_LiRPA` (PyTorch) and CORA (MATLAB), runtimes are reported per toolbox and relative to our enclosure within that toolbox to provide a fair comparison. At four layers, all methods are within $17\,\%$ of the sampled image and our enclosure is as tight as zonotope propagation. At eight layers, our enclosure is tighter than zonotope propagation and CROWN and within $5\,\%$ of α-CROWN, i.e., the tightness of our enclosure improves with depth while that of the set-propagation methods degrades. Computing our enclosure is $32\times$ faster than zonotope propagation, $256\times$ faster than CROWN, and $20\,600\times$ faster than α-CROWN, which additionally optimizes its bounds. Due to the monotone subnetworks, the IBP bounds coincide with ours, but our bounds are computed using two simple forward passes without any set propagation, which makes it $10\times$ to $11\times$ faster than IBP in `auto_LiRPA`.

5.3 Verification-Aware Training and the lower bound

To evaluate the effectiveness of our verification-aware training, we vary the width weight parameter β in the training loss ((5)) and compare the accuracy and enclosure tightness. Tab. 3 shows the

Table 3: Comparing the accuracy and enclosure tightness for varying width weights β in the loss function (5). Mean $\pm$ standard deviation over three seeds.

Benchmark	β	**Accuracy** $\uparrow$	**Tightness** $\downarrow$
Analytic	0	0.9846 ± 0.0007	1.821 ± 0.136
	0.01	0.9857 ± 0.0034	1.024 ± 0.009
	0.05	$\mathbf{0.9883} \pm 0.0012$	1.004 ± 0.003
	0.2	0.9803 ± 0.0012	$\mathbf{1.000} \pm 0.000$
	1.0	0.9392 ± 0.0121	$\mathbf{1.000} \pm 0.000$
Tanks	0	$\mathbf{0.9999} \pm 0.0000$	1.883 ± 0.134
	0.01	0.9998 ± 0.0001	$\mathbf{1.000} \pm 0.000$
	0.05	0.9985 ± 0.0006	$\mathbf{1.000} \pm 0.000$
	0.2	0.9852 ± 0.0035	$\mathbf{1.000} \pm 0.000$
	1.0	0.8646 ± 0.0121	$\mathbf{1.000} \pm 0.000$

Table 4: Comparing the enclosure tightness for different numbers of input dimensions. Mean $\pm$ standard deviation over five seeds.

#Input Dims	**Tightness** $\downarrow$	
	Zonotope	Ours
2	1.65 ± 0.01	$\mathbf{1.045} \pm 0.008$
20	1.77 ± 0.09	$\mathbf{1.073} \pm 0.009$
100	360 ± 31	$\mathbf{1.167} \pm 0.039$

results. Using the width weight parameter β we can tradeoff accuracy against enclosure tightness: A small value for the width weight parameter increases accuracy, while a larger value favors enclosure tightness.

From $\beta = 0.05$ on, the enclosure coincides with the sampled image on both benchmarks, so any further gain from a larger β comes from changing the represented function rather than its decomposition, which is why accuracy drops there.

5.4 Scalability

Due to mixed-monotonicity of the neural network, our enclosure computation does not require to explicitly consider approximation errors. Set propagation approaches suffer from the wrapping effect of approximation errors across layers, which prevents the scalability of these approaches. We demonstrate the superior scalability of our approach across different dimensions: (i) the number of input dimensions, (ii) the number of neural network layers, and (iii) closed-loop evaluations.

Number of Input Dimension To show the superior scalability across the number of input dimensions, we significantly increase the number of input dimensions of the synthetic benchmark and compare the enclosure tightness of our approach against the baseline using zonotope propagation. Tab. 4 shows the results: While the tightness of the baseline explodes for $n = 100$, our approach remains significantly tighter. Further results are provided in Tab. 8.

Number of Layers To showcase the scalability across the number of layers, we compare in Tab. 5 the tightness of our approach against the baseline using zonotope propagation for different number of neural network layers at a fixed perturbation radius $\epsilon = 0.05$. Additionally, we report the smallest input set radius (crossover radius) at which our enclosure is smaller than the corresponding zonotope enclosure. When increasing the number of layers the crossover radius becomes smaller. Additional results are in Appendix A.7.

Closed-loop Systems To further highlight the scalability of our approach, we compare our enclosure tightness in a closed-loop setting, where we evaluate the tightness of computed reachable sets of dynamical systems over time. To this end, we train a mixed-monotone neural network to simulate

Table 5: Comparing the enclosure tightness for different numbers of neural network layers. Mean $\pm$ standard deviation over ten seeds. The crossover radius is reported as median (range) over seeds; "–" indicates no crossover.

#Layers κ	**Tightness** $\downarrow$		**Crossover Radius** $\downarrow$ *[median (range)]*
	Zonotope	Ours	
2	$\mathbf{1.08} \pm 0.01$	5.92 ± 0.79	–
3	$\mathbf{1.11} \pm 0.02$	2.47 ± 0.72	1.00 (0.1–1.5)
4	$\mathbf{1.14} \pm 0.03$	1.56 ± 0.37	0.20 (0.03–0.5)
6	1.24 ± 0.12	$\mathbf{1.13} \pm 0.12$	0.025 (0.01–0.15)
8	1.38 ± 0.25	$\mathbf{1.19} \pm 0.18$	0.04 (0.01–0.075)

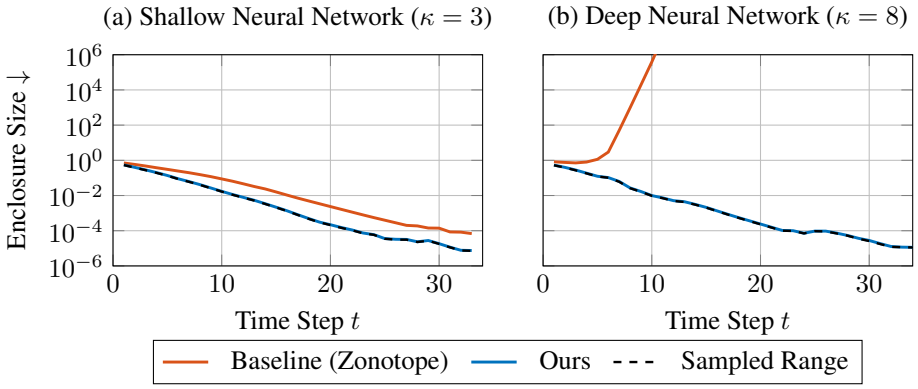


Figure 3: Enclosure size over time for closed-loop evaluations of the Tanks benchmark.

a time step of a dynamical system. We compare our approach against a baseline using standard, unconstrained feed-forward neural network and zonotope propagation. Fig. 3 plots the enclosure size over time: Our approach maintains tighter enclosures, which prevents the reachable set explosion of the baseline at $\kappa = 8$ neural network layers.

6 Conclusion

We introduce a neural network architecture that realizes mixed-monotone functions as the difference of two monotone subnetworks. The architecture replaces set propagation by two forward passes of opposite corners of the input interval, which makes computing an output enclosure as cheap as a point evaluation. Because the size of this enclosure is a closed-form function of the weights, we can directly minimize it during training, which turns the classical problem of finding a tight mixed-monotone decomposition into a learning problem; the monotone variation of the represented function gives a lower bound against which a learned decomposition is measured. Our evaluation shows that training reaches within $0.5\,\%$ of this lower bound at no cost in prediction accuracy, that computing our enclosure is orders of magnitude faster than CROWN and α-CROWN, and that on deep neural networks it is tighter than zonotope propagation and CROWN while staying within $5\,\%$ of α-CROWN. The advantage grows where verification is hardest: with the depth of the neural network, with the number of input dimensions, and under iteration in a closed loop.

Limitations Our enclosure bounds the neural network, not the system that the neural network approximates. On our sign-constrained architecture interval propagation is exact, so our enclosure coincides with IBP, but it computes $10\times$ to $11\times$ faster. The advantage of our approach over IBP is, that we can compute bounds via two forward passes without any propagation, which also makes the size is trainable. We restrict the decomposition to the separable form (2), which a general decomposition function could improve upon at the cost of enforcing the diagonal condition during training. Finally, we consider regression tasks, and the closed-loop results assume a cooperative plant under constant input; under state feedback the closed-loop map need not be monotone.

7 AI USE STATEMENT

In this work, we used generative AI tools for drafting sections, identifying and summarizing relevant literature, assisting during the implementation and in the writing of proofs, editing to improve readability of the paper, and creating or modifying TikZ figures or images. We have reviewed all AI-assisted work to ensure its correctness. We have carefully check the correctness of our AI-assisted implementation and our reported results are reproducible using the included supplementary material. We have checked every claim and reference against its source.

8 REPRODUCIBILITY STATEMENT

All benchmark definitions and evaluation details are included in Appendix A.2 to A.4. Additionally, we include a reproducibility package as supplementary material with this submission, which contains all source code and allows the reader to reproduce any results contained in the paper.

REFERENCES

Matthias Althoff. An introduction to CORA 2015. In *Proc. of the Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH)*, volume 34 of *EPiC Series in Computing*, pp. 120–151, 2015.

Matthias Althoff. Checking and establishing reachset conformance in CORA 2023. In *Proc. of the Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH)*, volume 96 of *EPiC Series in Computing*, pp. 9–33, 2023.

Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, M. Pawan Kumar, Jingyue Lu, and Pushmeet Kohli. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(42):1–39, 2020.

James A. Clarkson and Clifford R. Adams. On definitions of bounded variation for functions of two variables. *Transactions of the American Mathematical Society*, 35:824–854, 1933.

Donald L. Cohn. *Measure Theory*. Birkhäuser, Boston, 1993.

Samuel Coogan. Mixed monotonicity for reachability and safety in dynamical systems. In *IEEE Conference on Decision and Control (CDC)*, pp. 5074–5085, 2020.

Samuel Coogan and Murat Arcak. Efficient finite abstraction of mixed monotone systems. In *Hybrid Systems: Computation and Control (HSCC)*, pp. 58–67, 2015.

Hennie Daniels and Marina Velikova. Monotone and partially monotone neural networks. *IEEE Transactions on Neural Networks*, 21(6):906–917, 2010.

Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *Hybrid Systems: Computation and Control (HSCC)*, pp. 291–305, 2005.

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy A. Mann, and Pushmeet Kohli. Scalable verified training for provably robust image classification. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4841–4850, 2019.

Saber Jafarpour, Akash Harapanahalli, and Samuel Coogan. Interval reachability of nonlinear dynamical systems with neural network controllers. In *Learning for Dynamics and Control Conference (L4DC)*, volume 211 of *Proceedings of Machine Learning Research*, pp. 12–25, 2023.

Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification (CAV)*, pp. 97–117, 2017.

Mohammad Khajenejad and Sze Zheng Yong. Tight remainder-form decomposition functions with applications to constrained reachability and guaranteed state estimation. *IEEE Transactions on Automatic Control*, 68(12):7057–7072, 2023.

Ouail Kitouni, Niklas Nolte, and Mike Williams. Robust and provably monotonic networks. *Machine Learning: Science and Technology*, 4(3):035020, 2023.

Niklas Kochdumper and Matthias Althoff. Sparse polynomial zonotopes: A novel set representation for reachability analysis. *IEEE Transactions on Automatic Control*, 66(9):4043–4058, 2021.

Niklas Kochdumper, Christian Schilling, Matthias Althoff, and Stanley Bak. Open- and closed-loop neural network verification using polynomial zonotopes. In *NASA Formal Methods (NFM)*, pp. 16–36, 2023.

Lukas Koller, Tobias Ladner, and Matthias Althoff. Set-based training for neural network verification. *Transactions on Machine Learning Research*, 2025.

Lukas Koller, Tobias Ladner, and Matthias Althoff. Out of the shadows: Exploring a latent space for neural network verification. In *International Conference on Learning Representations (ICLR)*, 2026.

Tobias Ladner and Matthias Althoff. Automatic abstraction refinement in neural network verification using sensitivity analysis. In *Hybrid Systems: Computation and Control (HSCC)*, 2023.

Xingchao Liu, Xing Han, Na Zhang, and Qiang Liu. Certified monotonic neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020.

Pierre-Jean Meyer. Reachability analysis of neural networks using mixed monotonicity. *IEEE Control Systems Letters*, 6:3068–3073, 2022.

Mark Niklas Müller, Franziska Eckert, Marc Fischer, and Martin Vechev. Certified training: Small boxes are all you need. In *International Conference on Learning Representations (ICLR)*, 2023.

Davor Runje and Sharath M. Shankaranarayana. Constrained monotonic neural networks. In *International Conference on Machine Learning (ICML)*, pp. 29338–29353, 2023.

Joseph Sill. Monotonic networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 10, 1997.

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 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):41:1–41:30, 2019.

Aishwarya Sivaraman, Golnoosh Farnadi, Todd Millstein, and Guy Van den Broeck. Counterexample-guided learning of monotonic neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020.

Hal L. Smith. *Monotone Dynamical Systems: An Introduction to the Theory of Competitive and Cooperative Systems*, volume 41 of *Mathematical Surveys and Monographs*. American Mathematical Society, 1995.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. arXiv:2407.17032, 2024.

Manuel Wendl, Lukas Koller, Tobias Ladner, and Matthias Althoff. Training verifiably robust agents using set-based reinforcement learning. *Transactions on Machine Learning Research*, 2026.

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. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020.

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 *International Conference on Learning Representations (ICLR)*, 2021.

Liren Yang, Oscar Mickelin, and Necmiye Ozay. On sufficient conditions for mixed monotonicity. *IEEE Transactions on Automatic Control*, 64(12):5080–5085, 2019.

Cunliang Ye, Yongfu Wang, Yunlong Wang, and Ming Tie. Steering angle prediction for autonomous vehicles using neural networks. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 236(9):1991–2011, 2022.

Seungil You, David Ding, Kevin Canini, Jan Pfeifer, and Maya Gupta. Deep lattice networks and partial monotonic functions. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

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 (NeurIPS)*, volume 31, 2018.

A Appendix

A.1 Proofs

Proposition 3. *Given an input interval $\mathcal{X} = [\underline{x}, \overline{x}] \subseteq \mathbb{R}^{n_0}$, a neural network $\Phi = \Phi_+ - \Phi_-$, where Φ_+, Φ_- are monotonic on $\mathcal{X}$, then*

$$\mathbf{1}^\top \nu_\Phi(\mathcal{X}) \leq \|\texttt{enclose}(\Phi, \mathcal{X})\|_{\mathrm{I}}.$$

Moreover, for $\mathcal{X}$ there exists $\Phi' = \Phi'_+ - \Phi'_-$ with monotonic Φ'_+, Φ'_- such that equality holds.

Proof. For the lower bound, let $\{x_j\}_{j=0}^k \in \mathcal{S}$. Since $x_{j-1} \leq x_j$ and Φ_+, Φ_- are non-decreasing, the differences $\Phi_+(x_j) - \Phi_+(x_{j-1})$ and $\Phi_-(x_j) - \Phi_-(x_{j-1})$ are non-negative, and hence

$$|\Phi(x_j) - \Phi(x_{j-1})| \leq \Phi_+(x_j) - \Phi_+(x_{j-1}) + \Phi_-(x_j) - \Phi_-(x_{j-1})$$

for all $j \in [k]$. Summing over j, the right-hand side telescopes, and $\underline{x} \leq x_0$ and $x_k \leq \overline{x}$ give

$$\sum_{j=1}^{k} |\Phi(x_j) - \Phi(x_{j-1})| \leq \Phi_+(\overline{x}) - \Phi_+(\underline{x}) + \Phi_-(\overline{x}) - \Phi_-(\underline{x}).$$

The right-hand side does not depend on the sequence, so taking the supremum in (6) and multiplying by $\mathbf{1}^\top$ gives

$$\mathbf{1}^\top \nu_\Phi(\mathcal{X}) \leq \mathbf{1}^\top \big(\Phi_+(\overline{x}) - \Phi_-(\underline{x}) - \Phi_+(\underline{x}) + \Phi_-(\overline{x})\big) = \|\texttt{enclose}(\Phi, \mathcal{X})\|_{\mathrm{I}},$$

where the equality holds because the bounds of the enclosure (2) satisfy $\Phi_+(\underline{x}) - \Phi_-(\overline{x}) \leq \Phi_+(\overline{x}) - \Phi_-(\underline{x})$.

For the existence, let $x \in \mathcal{X}$. Applied to the interval $[\underline{x}, x] \subseteq \mathcal{X}$, (6) gives $\nu_\Phi([\underline{x}, x]) \leq \nu_\Phi(\mathcal{X})$, which is finite by the first part. For $x \leq x'$ in $\mathcal{X}$, appending x and x' to a sequence in $[\underline{x}, x]$ yields a sequence in $[\underline{x}, x']$ whose sum in (6) is larger by at least $|\Phi(x') - \Phi(x)|$; hence,

$$\nu_\Phi([\underline{x}, x']) - \nu_\Phi([\underline{x}, x]) \geq |\Phi(x') - \Phi(x)|.$$

Now let, for $x \in \mathcal{X}$,

$$\Phi'_+(x) = \tfrac{1}{2}\big(\nu_\Phi([\underline{x}, x]) + \Phi(x)\big), \qquad \Phi'_-(x) = \tfrac{1}{2}\big(\nu_\Phi([\underline{x}, x]) - \Phi(x)\big).$$

Then $\Phi' = \Phi'_+ - \Phi'_- = \Phi$ on $\mathcal{X}$, and since $|\Phi(x') - \Phi(x)| \pm \big(\Phi(x') - \Phi(x)\big) \geq \mathbf{0}$, the inequality above shows that Φ'_+ and Φ'_- are non-decreasing on $\mathcal{X}$. Every sequence in $[\underline{x}, \underline{x}]$ is constant, so $\nu_\Phi([\underline{x}, \underline{x}]) = \mathbf{0}$, and $[\underline{x}, \overline{x}] = \mathcal{X}$. As in the first part, the size of the enclosure of Φ' is therefore

$$\|\texttt{enclose}(\Phi', \mathcal{X})\|_{\mathrm{I}} = \mathbf{1}^\top \big(\Phi'_+(\overline{x}) - \Phi'_+(\underline{x}) + \Phi'_-(\overline{x}) - \Phi'_-(\underline{x})\big) = \mathbf{1}^\top \nu_\Phi(\mathcal{X}). \qquad \square$$

A.2 BENCHMARK DEFINITIONS

In this section, we define the analytic regression task and two dynamical systems. The two dynamical systems differ in whether they are cooperative. A system $\dot{x} = f(x, u)$ is cooperative if it satisfies the Kamke–Müller condition (Smith, 1995)

$$\frac{\partial f_{(i)}}{\partial x_{(j)}} \geq 0 \;\; \forall i \neq j, \qquad \frac{\partial f}{\partial u} \geq 0. \tag{7}$$

Under constant input, the successor map $x \mapsto x'$ of a cooperative system is non-decreasing, whereas the increment map $x \mapsto x' - x$ need not be, since $\partial x'_{(i)}/\partial x_{(i)} - 1$ may be negative.

Analytic Task The analytic task is the scalar

$$f(x) = \sin x + 0.3 \sin 3x + 0.05x \tag{8}$$

on $[-3, 3]$, chosen because its monotone variation is available in closed form.

Pendulum The pendulum system re-implements the Gymnasium `Pendulum-v1` dynamics (Towers et al., 2024), verified step-wise against the reference; the $\sin$ term makes it violate (7), so it is the non-cooperative case.

Tanks The tanks system is a cascaded two-tank system with Torricelli outflow,

$$\begin{aligned} \dot{x}_{(1)} &= -k_1\sqrt{x_{(1)}} + k_4 u, \\ \dot{x}_{(2)} &= k_2\sqrt{x_{(1)}} - k_3\sqrt{x_{(2)}}, \end{aligned} \tag{9}$$

integrated with a fourth-order Runge–Kutta scheme; it satisfies (7). On the discretised map used for training we verified numerically that the Jacobian of the successor map is entrywise non-negative, while the increment map is not non-decreasing. We therefore train on both targets and denote them *tanks* ($x' - x$) and *tanks-succ* (x'). Trajectories are split $70/15/15$ per episode, so no transition appears in two splits.

Dimension sweep The synthetic target is

$$f(x) = \sin x_{(1)} + 0.3 \sin 3x_{(2)} + \frac{0.05}{n} \sum_{i=1}^{n} x_{(i)}, \tag{10}$$

so that raising n enlarges the input without changing the intrinsic complexity. In the "all" regime, all inputs are perturbed; in the "local" regime, only $x_{(1)}$ and $x_{(2)}$ are, which keeps the input-box size independent of n as a control.

A.3 METRIC ESTIMATION

All enclosure sizes are evaluated on standardised outputs. For the tightness defined in Sec. 5, we evaluate the neural network at sampled inputs in $\mathcal{X}$ and construct the interval hull of the resulting outputs. Its interval norm provides a lower bound on the size of the exact image $\Phi(\mathcal{X})$; for positive estimates, the reported tightness is therefore an upper bound on its exact counterpart. Aggregation details are given in Appendix A.4, and runtime measurements in Appendix A.13 and A.14.

A.4 EXPERIMENTAL DETAILS

Comparability of the architectures. Our sign-constrained architecture is compared with approximately parameter-matched unconstrained and Lipschitz-constrained non-monotone networks. Within each experiment, all architectures share the data splits, optimiser settings and epoch budget, which we did not tune per architecture. Dynamical-system data are standardised with training-set statistics. In the verifier and depth–radius comparisons, all methods are applied to the same trained networks, with $\beta = 0$. For the export to `auto_LiRPA`, the verifier comparison retrains its networks with the same hyperparameters; since $\beta = 0$ leaves the decomposition unconstrained, our enclosures of these networks differ from those of the depth–radius networks at equal accuracy (Tab. 5 and 7). The dimension and closed-loop comparisons use separately trained networks, with $\beta = 0.05$ for the monotone architectures.

Optimiser and schedule. Adam with learning rate $2 \cdot 10^{-3}$, batch size 64, 80 to 150 epochs depending on the benchmark, no learning-rate decay, no early stopping. When $\beta > 0$ the weight is annealed in three stages over the epoch budget, from 0 to $\beta/2$ to β, so that the fit is established before the decomposition is tightened. The optimiser state is carried across stages.

Boxes seen by the width term. Each mini-batch unit consists of $B = 64$ training inputs followed by the $K = 8$ lower and $K = 8$ upper corners of boxes centred on randomly drawn training inputs, so a single forward pass serves both terms of (5). The radius is 0.2 in normalised input coordinates and the corners are clipped to the training domain. The boxes are drawn once per stage rather than per epoch, giving $\lfloor N/B \rfloor \cdot K$ distinct boxes that are then reused; for $N = 6000$ this is 744 boxes.

Initialisation of the sign-constrained blocks. Non-negative weights do not cancel, so the usual $1/\sqrt{n}$ scaling makes the forward pass grow like $\sqrt{n}$ per layer and diverge with depth. We use $|\mathcal{N}(0,1)| \cdot c/n$ with $c = 1.4$, which keeps the activation scale within $[0.3, 6]$ across the depths we evaluate. Outside roughly that window training fails: at $c = 5$ the network does not fit the analytic task at all. Each hidden layer splits its neurons in equal thirds over ReLU, $-\mathrm{ReLU}(-\cdot)$ and $\tanh$.

Seeds, sampling and aggregation. Prediction-accuracy and CORA depth–radius results use ten seeds, the `auto_LiRPA` and dimension sweeps five, and the width-weight and Lipschitz-budget sweeps three. The sampled reference for the tightness uses 800 samples per box and 150 boxes per radius, or 100 in the verifier and dimension comparisons. For one-step tightness ratios, enclosure sizes and denominator estimates are averaged over the boxes of a seed before dividing, and we report mean and standard deviation across seeds. A configuration passes the accuracy gate if every seed satisfies $R^2 \geq 0.9$; only passing configurations support comparative claims. Runtime conventions are given in Appendix A.13 and A.14.

Certification settings. Zonotope propagation uses CORA's default activation abstraction. In the iterated experiment the zonotope is reduced with Girard's method (Girard, 2005) to order 20 after every step; the sensitivity of the results to this order is reported in Appendix A.9.

Closed-loop rollouts. The closed-loop results of Sec. 5.4 use tanks-succ at $\kappa \in \{3, 8\}$, three seeds and twenty initial boxes per seed at $r_0 = 0.2$, with up to 40 steps under a constant input and 2000 sampled trajectories per box. At every step, the samples are checked to lie inside every computed enclosure up to a tolerance of 10^{-3} relative to the sampled width, and a violation aborts the experiment. A rollout ends when the sampled width falls below 10^{-5}, when propagation fails numerically, or at the horizon. Stepwise medians omit contracted rollouts after they end and count failed propagations as infinite widths.

Environment. Training and the sweeps were run on a Linux cluster. The CORA runtimes of Tab. 2 and Sec. 5.4 come from a single run on one otherwise idle machine with an Intel Core Ultra 9 285K (24 cores, 62 GB), MATLAB R2026a and CORA v2026.1.0. The `auto_LiRPA` timings of Tab. 7 were measured separately on an AMD Ryzen 7 Microsoft Surface Edition (Zen 2, 8 cores, 16 GB) with PyTorch restricted to eight threads. We report both absolute runtimes and runtime ratios within this measurement setup.

Reproducibility. The data-set seed, the train/validation/test split, the weight initialisation, the training stages, the evaluation boxes and the sampling reference are derived from a single per-repetition seed by fixed offsets, so that the streams do not overlap. Splits are per episode, not per sample.

A.5 Additional prediction-accuracy results

The unconstrained reference remains highly accurate at greater depth. The sign-constrained model remains close to it, while both Lipschitz-constrained models lose accuracy under the shared training protocol. Their degradation is not identical: at eight layers the monotone residual model has $R^2 = 0.752 \pm 0.048$ and the non-monotone model 0.635 ± 0.074. This comparison does not isolate all causes of the optimisation difficulty.

Table 6: Accuracy over depth at matched parameter count on the analytic task [*prediction accuracy* $R^2 \uparrow$], mean $\pm$ standard deviation over ten seeds. Bold marks the best architecture per depth. The gate applied elsewhere is $R^2 \geq 0.9$.

#Layers κ	**Architecture**			
	Baseline (unconstr.)	Ours (sign-constr)	Lip.-res. (monotone) Kitouni et al. (2023)	Lip. (non-monotone)
2	0.984 ± 0.002	$\mathbf{0.986} \pm 0.001$	0.973 ± 0.002	0.972 ± 0.002
3	$\mathbf{0.999} \pm 0.002$	0.992 ± 0.003	0.983 ± 0.001	0.983 ± 0.001
4	$\mathbf{1.000} \pm 0.000$	0.990 ± 0.005	0.982 ± 0.001	0.981 ± 0.001
6	$\mathbf{1.000} \pm 0.000$	0.992 ± 0.006	0.876 ± 0.022	0.838 ± 0.028
8	$\mathbf{1.000} \pm 0.000$	0.996 ± 0.004	0.752 ± 0.048	0.635 ± 0.074

Table 7: Tightness and runtime of all verifiers on the analytic task at $\epsilon = 0.05$ for all tested depths, on the same trained networks as Tab. 2. Tightness is mean $\pm$ standard deviation over five seeds; IBP has the same tightness as ours. Runtimes are relative to our enclosure within the same toolbox (`auto_LiRPA` for IBP, CROWN and α-CROWN; CORA for zonotope propagation); α-CROWN uses 20 optimisation iterations at learning rate 0.1. Bold marks the best tightness per row.

#Layers κ	**Tightness** $\downarrow$				**Runtime relative to ours** $\downarrow$			
	Ours / IBP	Zonotope	CROWN	α-CROWN	IBP	Zono.	CROWN	α-CR.
2	3.58 ± 0.55	1.08 ± 0.01	1.07 ± 0.01	$\mathbf{1.001} \pm 0.000$	10.8	6.5	23.5	1 490
3	2.00 ± 0.43	1.14 ± 0.03	1.12 ± 0.03	$\mathbf{1.004} \pm 0.002$	8.1	8.4	54.4	6 130
4	1.17 ± 0.12	1.16 ± 0.05	1.13 ± 0.04	$\mathbf{1.002} \pm 0.002$	9.9	11.7	112	8 750
6	1.07 ± 0.08	1.25 ± 0.10	1.18 ± 0.04	$\mathbf{1.002} \pm 0.001$	10.2	19.6	193	12 900
8	1.05 ± 0.08	1.44 ± 0.07	1.26 ± 0.06	$\mathbf{1.002} \pm 0.001$	11.2	31.8	256	20 600

A.6 Additional linear-relaxation results

Tab. 7 contains every depth in the auto_LiRPA comparison. CROWN is tighter than our enclosure at two and three layers, the means are close at four, and our enclosure is tighter at six and eight. α-CROWN stays close to the sampled reference throughout. α-CROWN uses a fixed 20-iteration optimisation budget. Export validation and timing conventions are given in Appendix A.14. IBP coincides with our enclosure on this architecture. With non-negative effective weights, an interval passed through a linear layer remains the layer evaluated at its two corners, and the monotone activations preserve this, so propagating an interval through Φ_+ and Φ_- and subtracting reproduces $\mathrm{enclose}(\Phi, \mathcal{X})$ without abstraction error at any layer. This follows from the sign constraints and not from the corner form: for the Lipschitz-residual (Kitouni et al., 2023), the IBP width exceeds our enclosure by a factor of 1.49 at two layers and 1.15 at eight ($\epsilon = 0.05$, ten seeds). We verified the equality numerically: in CORA both widths agree to the reported precision, and on the exported PyTorch model, where the subtraction is a fixed linear layer so that the branches are not bounded separately, they agree to a relative $1.1 \cdot 10^{-6}$ at worst and $6 \cdot 10^{-8}$ in median over the 245 network–radius pairs of the sweep.

A.7 Additional observations on the crossover

Between six and eight layers, the crossover-radius medians differ by one grid step and the observed seed ranges overlap (Tab. 5). These measurements do not resolve a further decrease in crossover radius at eight layers. Over the radius sweep at eight layers, our enclosure size per unit radius decreases from 1.448 to 1.219 across two orders of magnitude in radius, while that of zonotope propagation increases by a factor of twelve.

A.8 Additional width-weight and prediction-target results

From $\beta = 0$ to 0.05 on the analytic task, the tightness decreases from 1.82 to 1.00 (Tab. 3) while accuracy is unchanged, so the improvement comes from a tighter decomposition of the same function. Beyond this setting, the enclosure already coincides with the sampled image, so additional width reduction changes the represented function and reduces accuracy.

For tanks-succ, the ratio against the sampled range is 1.000 at the reported precision from $\beta = 0.01$ onward. This is empirical agreement with the sampled interval hull, not a separate formal proof that the trained decomposition is exactly optimal on every box. In practice, the width weight should be selected using both validation accuracy and enclosure quality, with tolerance for seed variability rather than requiring a rounded ratio to equal one exactly.

Prediction target. The reported comparison at depth eight holds the system, architecture and seeds fixed while switching the Tanks prediction target from the increment to the successor state. The excess tightness falls from $37\,\%$ to $8\,\%$ ($\epsilon = 0.05$, ten seeds), a reduction of about $78\,\%$. The successor-map structure permits a non-decreasing representation, whereas the increment map need not be non-decreasing. On Pendulum, prediction accuracy differs from the unconstrained baseline by at most 0.007 in R^2 at both depths (Tab. 1).

A.9 Iterated-certification details

We roll the certified set forward under a constant input,

$$\mathcal{X}_{t+1} = [d_\Phi(\underline{x}_t, \overline{x}_t), d_\Phi(\overline{x}_t, \underline{x}_t)], \tag{11}$$

with the constant input held fixed in each evaluation. The implementation used here cannot feed a zonotope back through the composite monotone layer across a horizon. We therefore compare the sign-constrained model with direct enclosure evaluation against an unconstrained model with zonotope propagation. This is an end-to-end comparison of model–verification pairings. The zonotope is propagated as a set across the horizon, reduced to a fixed order after every step and enclosed in a box only for reading the metric.

At three layers, no zonotope rollout becomes non-finite, while its tightness grows from 1.3 to 4.3 over 33 steps, peaking at 6.9. At eight layers, the median width exceeds 10^6 at step eleven and reaches $1.4 \cdot 10^{37}$ at step thirty; 26 of the 60 rollouts terminate with non-finite propagated sets, 41 exceed width 10^3 at some step, and 19 stay below width 10 throughout. Failures are carried at $+\infty$ when computing population statistics, as detailed in Appendix A.4. The horizon covers a contraction of the sampled reference by more than four orders of magnitude.

To test whether this divergence is an artefact of the reduction order, the eight-layer unconstrained networks of the closed-loop experiment are rolled out on the successor maps of Tanks and Pendulum from five initial boxes per seed at $r_0 = 0.2$, with 800 samples per box and orders 10, 20, 50, 100 and 200. We define the onset of divergence as the first step at which the zonotope enclosure exceeds 100 times the size of the sampled reference. Raising the order from 20 to 200 lowers the width at step 20 by about one order of magnitude, with little change between 100 and 200, while the median onset of divergence does not change on either system.

These results concern the learned models; soundness for a physical plant would also require accounting for model error.

A.10 Additional input-dimension results

Tab. 8 reports both perturbation regimes and all tested input dimensions. In the "local" regime, the input-box size remains near 1.9 and the sign-constrained model's tightness remains near one. In the "all" regime, both the input-box size and its tightness increase with dimension. Configurations that fail the prediction accuracy gate are marked and do not support matched-accuracy claims. In CORA, our enclosure takes 13–16 μs per box across all tested dimensions, while zonotope propagation of the unconstrained model takes 4.5 to 7.5 times as long, increasing with n (Appendix A.13).

Table 8: Scaling in the input dimension at $\epsilon = 0.5$, mean $\pm$ standard deviation over five seeds, [*Tightness* $\downarrow$], on a synthetic target whose intrinsic complexity is held fixed as n grows (only two inputs enter nonlinearly, see Appendix A.2), so that raising n enlarges the certification problem without enlarging the learning problem. $\|\mathcal{X}\|_{\mathrm{I}}$ is the ℓ_1-diameter of the perturbation set: it grows with n in the *all* regime and does not in the *local* one. The monotone architectures are certified by (2), the unconstrained one by zonotope propagation. Daggers mark configurations that fail the R^2 gate; they are shown for completeness and carry none of our claims.

Regime	**#Input Dims**	**Input Set Size** $\|\mathcal{X}\|_{\mathrm{I}}$	**Ours**	**Lip.-res.** Kitouni et al. (2023)	**Zonotope**
all	2	1.92	1.045 ± 0.008	19.4 ± 0.4	1.65 ± 0.01
	5	4.82	1.054 ± 0.012	38.2 ± 3.2	1.59 ± 0.05
	10	9.56	1.050 ± 0.009	$86.5^{\dagger} \pm 6.9$	1.66 ± 0.08
	20	19.18	1.073 ± 0.009	$181^{\dagger} \pm 37$	1.77 ± 0.09
	50	47.90	1.105 ± 0.013	$378^{\dagger} \pm 38$	$152^{\dagger} \pm 12$
	100	95.91	1.167 ± 0.039	$862^{\dagger} \pm 95$	$360^{\dagger} \pm 31$
local	2	1.92	1.045 ± 0.008	19.4 ± 0.4	1.65 ± 0.01
	5	1.93	1.047 ± 0.011	16.1 ± 1.3	1.54 ± 0.05
	10	1.90	1.041 ± 0.006	$19.3^{\dagger} \pm 1.8$	1.55 ± 0.07
	20	1.91	1.045 ± 0.008	$20.3^{\dagger} \pm 4.9$	1.52 ± 0.05
	50	1.91	1.040 ± 0.005	$16.8^{\dagger} \pm 1.8$	$2.42^{\dagger} \pm 0.16$
	100	1.92	1.028 ± 0.007	$18.9^{\dagger} \pm 2.0$	$2.60^{\dagger} \pm 0.06$

A.11 Enforcing non-negative weights

The sign-constrained construction requires non-negative weights in the forward pass. We use $|W|$ and optimise the unconstrained parameters W. For nonzero entries of W, the gradient of the training objective is obtained by the chain rule:

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial |W|} \odot \operatorname{sign}(W). \tag{12}$$

At zero, where the absolute value is not differentiable, we choose the subgradient $+1$ for the absolute-value operation. The weights used in the forward pass are therefore non-negative throughout training, preserving the monotonicity of both blocks.

An alternative is to project the weights after each update, $W \leftarrow \max(W, 0)$. In our experiments, projected updates led to an increasing fraction of zero weights (Appendix A.12), motivating our use of the absolute-value parametrisation.

A.12 Projected Training and Dead Weights

Projected updates clip a weight that reaches the boundary to exactly zero; momentum then drives it into the infeasible half-space again and the next projection repeats the clip. On the analytic task at width 32 we observed 28 to 31 % of the monotone weights in the infeasible half-space at each projection interval, and after 120 epochs 32.3 % of all monotone weights (703 of 2176) were exactly zero, with R^2 falling from 0.98 to 0.76. The damage grows with the number of projections rather than with the architecture: the same configuration trained for 40 epochs with two projections still reached $R^2 = 0.977$ to 0.984 over five seeds.

With the reparametrisation (12) the same configuration trained for 120 epochs gives $R^2 = 0.9856$ with no weight exactly zero; 28.3 % of the raw weights are negative, which is harmless since only $|W|$ enters the forward pass. The analytic gradient of (12) agrees with central differences to a relative error of $2.5 \cdot 10^{-10}$.

A.13 Timing Details

The CORA runtimes of Tab. 2 and Sec. 5.4 stem from one run on an Intel Core Ultra 9 285K (MATLAB R2026a, CORA v2026.1.0) with 200 boxes per configuration and the minimum over 25

repetitions after a warm-up evaluation; the interquartile range lies within $10\,\%$ of the median for our enclosure and within $20\,\%$ for zonotope propagation, whereas means are dominated by isolated scheduling outliers. The networks are freshly initialised, since the runtime does not depend on the weights.

A.14 Export to auto_LiRPA

The exported model reproduces the CORA network layer by layer in PyTorch rather than through an ONNX round-trip, since the architecture is a nest of composite layers that such an export handles poorly. Each monotone block is a stack of κ linear layers whose weight matrices enter the forward pass as $\left|W^{(k)}\right|$, with every hidden layer split into equal thirds over ReLU, $-\mathrm{ReLU}(-\cdot)$ and $\tanh$ (Appendix A.4); the $2m$ blocks are evaluated in parallel and the graph ends in a fixed linear layer with weights $[+I\ -I]$, so that the subtraction $\Phi = \Phi_+ - \Phi_-$ is part of the graph. This is what the comparison requires: if the two branches were bounded separately and subtracted afterwards, the result would reproduce $\|\texttt{enclose}(\Phi, \mathcal{I})\|_{\mathrm{I}}$ by construction and the comparison would measure nothing. Weights are transferred from the MATLAB checkpoints without retraining, and the comparison covers the sign-constrained networks of the analytic benchmark at hidden width 32, $\kappa \in \{2, 3, 4, 6, 8\}$ and five seeds each.

Two assertions guard the export and abort the run on failure. The rebuilt forward pass must match the CORA network over 512 probe points per network drawn uniformly from $[-1, 1]^n$, and interval propagation in `auto_LiRPA` must reproduce $\|\texttt{enclose}(\Phi, \mathcal{I})\|_{\mathrm{I}}$, as argued in Appendix A.6. Both are checked in single precision end to end; over the 245 network–radius pairs of the sweep the second agrees to a relative $1.1 \cdot 10^{-6}$ at worst and $6 \cdot 10^{-8}$ in median. We use `auto_LiRPA` v0.7.2 with PyTorch 2.11.0 (CPU build) under Python 3.11, with 20 optimisation iterations at α learning rate 0.1, stated explicitly because the library defaults differ between versions.

Two conventions govern the timings of Tab. 7, which are the fastest of ten repetitions after a warm-up. The `BoundedModule` is constructed once per network and excluded from the measurement, since a verification pipeline builds the graph once and queries many boxes; counting it would make the reported factor a function of how many boxes happen to be evaluated. And `auto_LiRPA`'s own interval propagation, which on this architecture computes the same number as the evaluation at the two corners, is eight to eleven times slower than it across every depth and radius we measure. This indicates an implementation-overhead contribution even before linear-relaxation optimisation is considered. The measurements do not separately identify framework overhead and the cost of the bound algorithm.