Journal of Machine Learning Research 26 (2025) 1-44 Submitted 8/24; Revised 6/25; Published 7/25

PREMAP: A Unifying PREiMage APproximation Framework for Neural Networks

Xiyue Zhang[*†] XIYUE.ZHANG@CS.OX.AC.UK
Benjie Wang[*] BENJIE.WANG@CS.OX.AC.UK
Marta Kwiatkowska MARTA.KWIATKOWSKA@CS.OX.AC.UK
Department of Computer Science
University of Oxford
Oxford, OX1 3QD, UK

Huan Zhang HUAN@HUAN-ZHANG.COM
Department of Electrical and Computer Engineering
University of Illinois Urbana–Champaign
Urbana, IL 61801, USA

Editor: Qiang Liu

Abstract

Most methods for neural network verification focus on bounding the *image*, i.e., set of outputs for a given input set. This can be used to, for example, check the robustness of neural network predictions to bounded perturbations of an input. However, verifying properties concerning the *preimage*, i.e., the set of inputs satisfying an output property, requires abstractions in the input space. We present a general framework for preimage abstraction that produces under- and over-approximations of any polyhedral output set. Our framework employs cheap parameterised linear relaxations of the neural network, together with an anytime refinement procedure that iteratively partitions the input region by splitting on input features and neurons. The effectiveness of our approach relies on carefully designed heuristics and optimisation objectives to achieve rapid improvements in the approximation volume. We evaluate our method on a range of tasks, demonstrating significant improvement in efficiency and scalability to high-input-dimensional image classification tasks compared to state-of-the-art techniques. Further, we showcase the application to quantitative verification and robustness analysis, presenting a sound and complete algorithm for the former and providing sound quantitative results for the latter.

Keywords: preimage approximation, abstraction and refinement, linear relaxation, formal verification, neural network

1. Introduction

Despite the remarkable empirical success of neural networks, ensuring their safety against potentially adversarial behaviour, especially when using them as decision-making components in autonomous systems (Bojarski et al., 2016; Codevilla et al., 2018; Yun et al., 2017), is an important and challenging task. Towards this aim, various approaches have been de-

∗. Equal contribution
†. Current address: School of Computer Science, University of Bristol, UK (Correspondence to X. Zhang, xiyue.zhang@bristol.ac.uk)

veloped for the verification of neural networks, with extensive effort devoted, in particular, to the problem of *local robustness verification*, which focuses on deciding the presence or absence of adversarial examples (Szegedy et al., 2013; Biggio et al., 2013) within an ϵ-perturbation neighbourhood (Huang et al., 2017; Katz et al., 2017; Zhang et al., 2018; Bunel et al., 2018; Tjeng et al., 2019; Singh et al., 2019; Xu et al., 2020, 2021; Wang et al., 2021b).

While local robustness verification is useful for certifying that a neural network has the same prediction in a neighbourhood of an input, it does not provide finer-grained information on the behaviour of the network in the input domain. An alternative and more general approach for neural network analysis is to construct the *preimage* abstraction of its predictions (Matoba and Fleuret, 2020; Dathathri et al., 2019). Given a set of outputs, the preimage is defined as the set of all inputs mapped by the neural network to that output set. For example, given a particular action for a neural network controller (e.g., drive left), the preimage captures the set of percepts (e.g., car positions) that cause the neural network to take this action. By characterising the preimage symbolically in an abstract representation, e.g., polyhedra, one can perform more complex analysis for a wider class of properties beyond local robustness, such as computing the *proportion* of inputs satisfying a property (Webb et al., 2019b; Mangal et al., 2019), or performing downstream reasoning tasks.

Unfortunately, exact preimage generation (Matoba and Fleuret, 2020) is intractable at scale, as it requires splitting into input subregions where the neural network is linear. Each such subregion corresponds to a set of determined activation patterns of the nonlinear neurons, the number of which grows exponentially with the number of neurons. Therefore, we focus on the problem of *preimage approximation*, that is, constructing symbolic abstractions for the preimage. In this work, we propose PREMAP, a general framework for preimage approximation that computes under-approximations and over-approximations represented as disjoint unions of polytopes (DUP).

Our method leverages recent progress in local robustness verification, which uses parameterised linear relaxations of neural networks together with divide-and-conquer refinement strategies to analyse the input space in an efficient and GPU-friendly manner (Zhang et al., 2018; Wang et al., 2021b). We observe that, unlike robustness verification, where the goal is to determine the behaviour of the neural network at the *worst-case* point in the input space (and thus verify or falsify the property), in preimage approximation we instead aim to minimize the *overall* difference in volume between the approximation and the (intractable) exact preimage. Thus, we design a methodology that focuses on effectively optimising this new volume-based objective, while maintaining the GPU parallelism, efficiency, and flexibility drawn from the state-of-the-art robustness verifiers.

In more detail, this paper makes the following novel contributions:

1. the first unifying framework capable of efficiently generating symbolic under- and over-approximations of the preimage abstraction of any polyhedron output set;

2. an efficient and anytime preimage refinement algorithm, which iteratively partitions the input region into subregions using input and/or intermediate (ReLU) splitting (hyper)planes;

3. carefully-designed heuristics for selecting input features and neurons to split on, which (i) take advantage of GPU parallelism for efficient evaluation; and (ii) significantly improve approximation quality compared to naïve baselines;

4. a novel differentiable optimisation objective for improving preimage approximation precision, with respect to (i) convex bounding parameters of nonlinear neurons and (ii) Lagrange multipliers for neuron splitting constraints;

5. empirical evaluation of preimage approximation on a range of datasets, and an application to the problem of quantitative verification;

6. a publicly-available software implementation of our preimage approximation framework (Zhang et al., 2025).

This work significantly extends the preliminary version in Zhang et al. (2024b) in the following ways: (i) introducing an *over-approximation* algorithm within the framework, with accompanying empirical results; (ii) improving the refinement procedure through new heuristics for selecting input features to split on, using only 49.7% (avg.) computation time of the prior method to achieve the same precision (Sections 5.3, 6.2.2); (iii) introducing Lagrangian relaxation to enforce neuron splitting constraints, enabling further optimisation of the approximations with precision gains of up to 58.6% (avg.) for a MNIST preimage approximation task (Sections 5.5, 6.2.3); and (iv) an extended empirical evaluation of the framework.

The paper is organized as follows. We present related works in Section 2. Section 3 introduces the notation and preliminary definitions of neural networks, linear relaxation and polyhedra representations. In Section 4, we present the formulation of the problems studied, namely preimage approximation and quantitative analysis of neural networks. Our preimage approximation method is provided in Section 5, together with the application to quantitative verification of neural networks and proofs of soundness and completeness. In Section 6, we present the experimental evaluation of our approach and demonstrate its effectiveness and scalability compared to the state-of-the-art techniques, and applications in quantitative verification and robustness analysis. We conclude the paper in Section 7.

2. Related Work

Our paper is related to a series of works on robustness verification of neural networks. To address the scalability issues with *complete* verifiers (Huang et al., 2017; Katz et al., 2017; Tjeng et al., 2019) based on constraint solving, convex relaxation (Salman et al., 2019) has been used for developing highly efficient *incomplete* verification methods (Zhang et al., 2018; Wong and Kolter, 2018; Singh et al., 2019; Xu et al., 2020). Later works employed the branch-and-bound (BaB) framework (Bunel et al., 2018, 2020) to achieve completeness, using incomplete methods for the bounding procedure (Xu et al., 2021; Wang et al., 2021b; Ferrari et al., 2022). In this work, we adapt convex relaxation for efficient preimage approximation. Further, our divide-and-conquer procedure is analogous to BaB, but focuses on maximising covered volume for under-approximation (resp. minimising for over-approximation) rather than maximising or minimising a function value.

There are also works that have sought to define a weaker notion of local robustness known as *statistical robustness* (Webb et al., 2019b; Mangal et al., 2019; Wang et al., 2021a), which requires that a proportion of points under some perturbation distribution around an input point are classified in the same way. Verification of statistical robustness is typically achieved by sampling and statistical guarantees (Webb et al., 2019b; Baluta et al., 2021; Tit et al., 2021; Yang et al., 2021). In this paper, we apply our symbolic approximation approach to quantitative analysis of neural networks, while providing *exact quantitative* rather than *statistical* evaluation (Webb et al., 2019a). In particular, similarly to Xiang et al. (2020); Rober et al. (2023), while we employ sampling in order to guide our divide-and-conquer procedure, the guarantees obtained are exact.

Another line of related works considers deriving exact or approximate abstractions of neural networks, which are applied for explanation (Sotoudeh and Thakur, 2021), verification (Elboher et al., 2020; Pulina and Tacchella, 2010), reachability analysis (Prabhakar and Afzal, 2019), and preimage approximation (Dathathri et al., 2019; Kotha et al., 2023). Dathathri et al. (2019) leverages symbolic interpolants (Albarghouthi and McMillan, 2013) for preimage approximations, facing exponential complexity in the number of hidden neurons. Kotha et al. (2023) considers the preimage over-approximation problem via inverse bound propagation, but their approach cannot be directly extended to the under-approximation setting. They also do not consider any strategic branching and refinement methodologies like those in our unified framework. Our anytime algorithm, which combines convex relaxation with principled splitting strategies for refinement, is applicable for both under- and over-approximations.

In the context of analysis of systems with neural network controllers, the backward reachability problem is to compute the set of states for which a system's trajectories can reach a particular target region within a finite time horizon. Prior works have explored both exact computation of this set (Vincent and Schwager, 2021) as well as guaranteed over-approximation (Rober et al., 2022, 2023; Zhang et al., 2023, 2024a; Kotha et al., 2023). Empirically, when applied to neural network controllers, we find that our approach performs competitively with the state-of-the-art method of Kotha et al. (2023).

3. Preliminaries

We use $f : \mathbb{R}^d \to \mathbb{R}^m$ to denote a feed-forward neural network. For layer i, we use $\mathbf{W}^{(i)}$ to denote the weight matrix, $\mathbf{b}^{(i)}$ the bias, $\mathbf{z}^{(i)}$ the pre-activation neurons, and $\hat{\mathbf{z}}^{(i)}$ the post-activation neurons, such that we have $\mathbf{z}^{(i)} = \mathbf{W}^{(i)}\hat{\mathbf{z}}^{(i-1)} + \mathbf{b}^{(i)}$. We use $h^{(i)}(x)$ to denote the function from input to pre-activation neurons, and $a^{(i)}(x)$ the function from input to the post-activation neurons, i.e., $\mathbf{z}^{(i)} = h^{(i)}(x)$ and $\hat{\mathbf{z}}^{(i)} = a^{(i)}(x)$. In this paper, we focus on ReLU neural networks with $a^{(i)}(x) = \mathrm{ReLU}(h^{(i)}(x))$, where $\mathrm{ReLU}(h) := \max(h, 0)$ is applied element-wise. However, our method can be generalized to other activation functions that can be bounded by linear functions, similarly to Zhang et al. (2018).

Linear Relaxation of Neural Networks. Nonlinear activation functions lead to the NP-completeness of the neural network verification problem as proved in Katz et al. (2017). To address such intractability, linear relaxation is often used to transform the nonconvex constraints into linear programs. As shown in Figure 1, given *concrete* lower and upper bounds $\mathbf{l}^{(i)} \leq h^{(i)}(x) \leq \mathbf{u}^{(i)}$ on the pre-activation values of layer i, there are

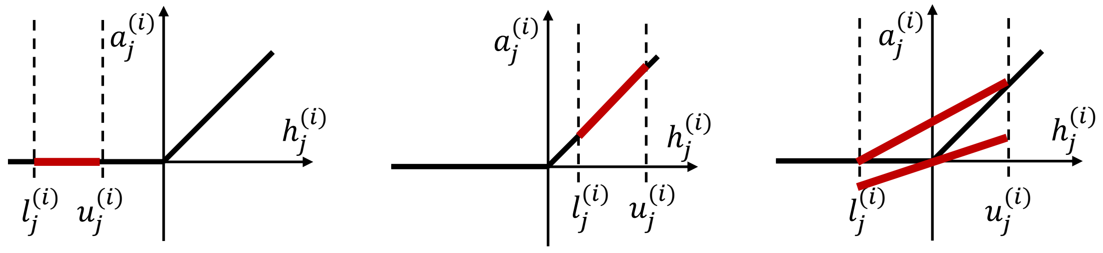


Figure 1: Linear bounding functions for inactive, active, unstable ReLU neurons.

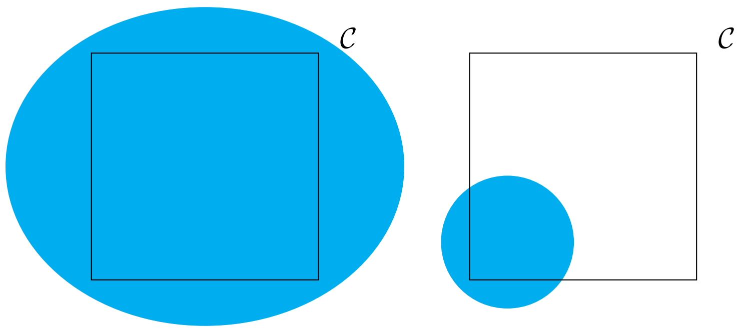
three cases to consider. In the *inactive* ($u_j^{(i)} \leq 0$) and *active* ($l_j^{(i)} \geq 0$) cases, the mapping function from pre-activation to post-activation values becomes linear, with $a_j^{(i)}(x) = 0$ and $a_j^{(i)}(x) = h_j^{(i)}(x)$ respectively. In the *unstable* case, $a_j^{(i)}(x)$ can be bounded by

$$\alpha_j^{(i)} h_j^{(i)}(x) \leq a_j^{(i)}(x) \leq -\frac{u_j^{(i)} l_j^{(i)}}{u_j^{(i)} - l_j^{(i)}} + \frac{u_j^{(i)}}{u_j^{(i)} - l_j^{(i)}} h_j^{(i)}(x) \tag{1}$$

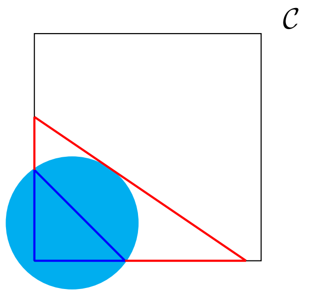
where $\alpha_j^{(i)}$ is a configurable parameter that produces a valid lower bound for any value in $[0, 1]$. Linear bounds can also be obtained for other non-piecewise linear activation functions by considering the characteristics of the activation function, such as the S-shape activation functions (Zhang et al., 2018; König et al., 2024).

Linear relaxation can be used to compute linear lower and upper bounds of the form $\underline{\mathbf{A}}x + \underline{\mathbf{b}} \leq f(x) \leq \overline{\mathbf{A}}x + \overline{\mathbf{b}}$ on the output of a neural network, for a given bounded input region $\mathcal{C}$. These methods are known as linear relaxation based perturbation analysis (LiRPA) algorithms (Xu et al., 2020, 2021; Singh et al., 2019). In particular, *backward-mode* LiRPA computes linear bounds on f by propagating linear bounding functions backward from the output, layer by layer, to the input layer.

Polytope Representations. Given an Euclidean space $\mathbb{R}^d$, a polyhedron T is defined to be the intersection of a finite number of half spaces. More formally, suppose we have a set of linear constraints defined by $\psi_i(x) := c_i^T x + d_i \geq 0$ for $i = 1, ...K$, where $c_i \in \mathbb{R}^d, d_i \in \mathbb{R}$ are constants, and $x = (x_1, ..., x_d)$ is a tuple of variables. Then a polyhedron is defined as $T = \{x \in \mathbb{R}^d | \bigwedge_{i=1}^K \psi_i(x)\}$, where T consists of all values of x satisfying the first-order logic (FOL) formula $\alpha(x) := \bigwedge_{i=1}^K \psi_i(x)$. We use the term polytope to refer to a bounded polyhedron, that is, a polyhedron T such that $\exists R \in \mathbb{R}^{>0} : \forall x_1, x_2 \in T$, $\|x_1 - x_2\|_2 \leq R$ holds. The abstract domain of polyhedra has been widely used for the verification of neural networks and computer programs as in Singh et al. (2019); Benoy (2002); Boutonnet and Halbwachs (2019). An important type of polytope is the hyperrectangle (box), which is a polytope defined by a closed and bounded interval $[\underline{x_i}, \overline{x_i}]$ for each dimension, where $\underline{x_i}, \overline{x_i} \in \mathbb{Q}$. More formally, using the linear constraints $\phi_i := (x_i \geq \underline{x_i}) \wedge (x_i \leq \overline{x_i})$ for each dimension, the hyperrectangle takes the form $\mathcal{C} = \{x \in \mathbb{R}^d | x \models \bigwedge_{i=1}^d \phi_i\}$, where $x \models \bigwedge_{i=1}^d \phi_i$ denotes that the input x satisfies the constraints specified by the conjunction of inequalities.



(a) Standard NN Verification: decide whether $f(x) \in O, \forall x \in \mathcal{C}$.



(b) NN Preimage Approximation: characterize $f^{-1}(O) := \{x \in \mathcal{C} \mid f(x) \in O\}$.

Figure 2: Illustration of the preimage approximation problem. In contrast to NN robustness verification, where the goal is to answer `Yes` or `No` for the statement $f(x) \in O, \forall x \in \mathcal{C}$, in preimage approximation the goal is to find a bounding under-approximation — and over-approximation — to the preimage $f^{-1}(O)$. ● indicates the region where $f(x) \in O$.

4. Problem Formulation

4.1 Preimage Approximation

In this work, we are interested in the problem of computing preimages for neural networks. Given a subset $O \subset \mathbb{R}^m$ of the codomain, the preimage of a function $f : \mathbb{R}^d \to \mathbb{R}^m$ is defined to be the set of all inputs $x \in \mathbb{R}^d$ that are mapped to an element of O by f. For neural networks in particular, the input is typically restricted to some bounded input region $\mathcal{C} \subset \mathbb{R}^d$. In this work, we restrict the output set O to be a polyhedron, and the input set $\mathcal{C}$ to be an axis-aligned hyperrectangle region $\mathcal{C} \subset \mathbb{R}^d$, as these are commonly used in neural network verification.

As illustrated in Figure 2, we contrast the preimage approximation problem with the setting of neural network robustness verification, where the goal is to answer a binary question, i.e., whether $f(x) \in O$ holds for all $x \in \mathcal{C}$. In preimage approximation, the objective is to compute the explicit characterisation of the preimage $f^{-1}(O)$ for the targeted output set O. In particular, we aim to compute bounding under- and over-approximations (depicted by — and —, respectively) of the true preimage (illustrated by ●) within the input domain. We now define the notion of a restricted preimage.

Definition 1 (Restricted Preimage) *Given a neural network $f : \mathbb{R}^d \to \mathbb{R}^m$, and an input set $\mathcal{C} \subset \mathbb{R}^d$, the restricted preimage of an output set $O \subset \mathbb{R}^m$ is defined to be the set $f_{\mathcal{C}}^{-1}(O) := \{x \in \mathbb{R}^d | f(x) \in O \wedge x \in \mathcal{C}\}$.*

Example 1 *To illustrate our problem formulation and approach, we introduce a vehicle parking task from Ayala et al. (2011) as a running example. In this task, there are four parking lots, located in each quadrant of a 2×2 grid $[0, 2]^2$, and a neural network with two hidden layers of 10 ReLU neurons $f : \mathbb{R}^2 \to \mathbb{R}^4$ is trained to classify which parking lot an input point belongs to. To analyze the behaviour of the neural network in the input region $[0, 2] \times [0, 2]$, we set $\mathcal{C} = \{x \in \mathbb{R}^2 | (0 \leq x_1 \leq 2) \wedge (0 \leq x_2 \leq 2)\}$. Then the restricted* *preimage $f_{\mathcal{C}}^{-1}(O)$ of the set $O = \{y \in \mathbb{R}^4 | \bigwedge_{i \in \{2,3,4\}} y_1 - y_i \geq 0\}$ is the subspace of the region $[0,2] \times [0,2]$ that is labelled as parking lot 1 by the neural network.*

We focus on *provable* approximations of the preimage. Given a first-order formula A, α is an *under-approximation* (resp. *over-approximation*) of A if it holds that $\forall x.\alpha(x) \implies A(x)$ (resp. $\forall x.A(x) \implies \alpha(x)$). In our context, the restricted preimage is defined by the formula $A(x) = (f(x) \in O) \wedge (x \in \mathcal{C})$, and we restrict to approximations α that take the form of a disjoint union of polytopes (DUP). The goal of our method is to generate a DUP approximation $\mathcal{T}$ that is as tight as possible; that is, we aim to maximise the volume $\text{vol}(\mathcal{T})$ of an under-approximation, or minimise the volume $\text{vol}(\mathcal{T})$ of an over-approximation.

Definition 2 (Disjoint Union of Polytopes) *A disjoint union of polytopes (DUP) is a FOL formula α of the form $\alpha(x) := \bigvee_{i=1}^{D} \alpha_i(x)$, where each α_i is a polytope formula (conjunction of a finite set of linear half-space constraints), with the property that $\alpha_i \wedge \alpha_j$ is unsatisfiable for any $i \neq j$.*

4.2 Quantitative Properties

One of the most important verification problems for neural networks is that of proving guarantees on the output of a network for a given input set (Gehr et al., 2018; Gopinath et al., 2020; Ruan et al., 2018). This is often expressed as a property of the form (I, O) such that $\forall x \in I \implies f(x) \in O$. We can generalize this to *quantitative* properties:

Definition 3 (Quantitative Property) *Given a neural network $f : \mathbb{R}^d \to \mathbb{R}^m$, a measurable input set with non-zero measure (volume) $I \subseteq \mathbb{R}^d$, a measurable output set $O \subseteq \mathbb{R}^m$, and a rational proportion $p \in [0,1]$, we say that the neural network satisfies the property (I, O, p) if $\frac{\text{vol}(f_I^{-1}(O))}{\text{vol}(I)} \geq p$.*

Note that the restricted preimage of a polyhedron under a neural network is Lebesgue measurable since polyhedra (intersection of a finite number of half-spaces) are Borel measurable and NNs are continuous functions.

Example 2 *Consider the vehicle parking task, where the goal is to predict where to park among four parking lots. Consider the input region $I = \{x \in \mathbb{R}^2 \mid x \in [0,1]^2\}$, representing the first parking lot region, and the output set $O = \{y \in \mathbb{R}^4 | \bigwedge_{i=2}^{4} y_1 - y_i \geq 0\}$, which specifies the neural network predicting that the vehicle should park in the first lot, i.e., y_1 is the largest score among all decisions. Let the quantitative proportion be $p = 0.9$. This defines a quantitative property (I, O, p) asserting that the volume of the generated preimage under-approximation $f_I^{-1}(O)$, from which the neural network maps to O, is at least 90% of the total volume of I.*

Neural network verification algorithms can be characterised by two main properties: *soundness*, which states that the algorithm always returns correct results, and *completeness*, which states that the algorithm always reaches a conclusion on any verification query (Liu et al., 2021). We now define the soundness and completeness of verification algorithms for quantitative properties.

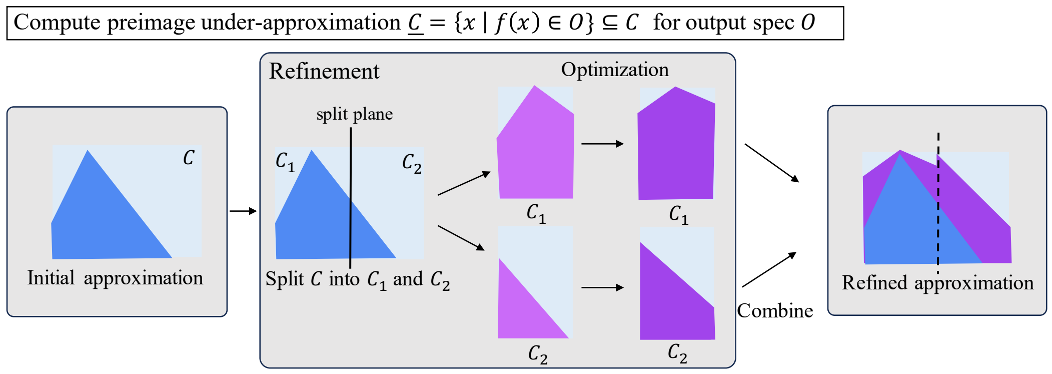

Compute preimage under-approximation $\underline{C} = \{x \mid f(x) \in O\} \subseteq C$ for output spec O
Initial approximation
C
Refinement
split plane
C_1
C_2
Split C into C_1 and C_2
Optimization
C_1
C_2
Combine
Refined approximation

Figure 3: Illustration of the workflow for preimage under-approximation (shown in 2D for clarity). Given a neural network $f : \mathbb{R}^d \to \mathbb{R}^m$ and output specification $O \subset \mathbb{R}^m$, our algorithm generates an under-approximation $\underline{\mathcal{C}}$ to the preimage in the input region $\mathcal{C}$. Starting from the input region $\mathcal{C}$, the procedure repeatedly splits the selected region into smaller subregions $\mathcal{C}_1$ and $\mathcal{C}_2$ with tighter input bounds, and then optimises the bounding and Lagrangian parameters to increase the volume and thus improve the quality of the under-approximation. The refined under-approximation is combined into a union of polytopes.

Definition 4 (Soundness) *A verification algorithm QV is sound if, whenever QV outputs* True*, the property (I, O, p) holds.*

Definition 5 (Completeness) *A verification algorithm QV is complete if (i) QV never returns* Unknown*, and (ii) whenever QV outputs* False*, the property (I, O, p) does not hold.*

If the property (I, O) holds, then the quantitative property $(I, O, 1)$ holds, while quantitative properties for $0 \leq p < 1$ provide more information when (I, O) does *not* hold. Most neural network verification methods produce approximations of the *image* of I in the output space, which cannot be used to verify quantitative properties. Preimage *over-approximations* include points outside of the true preimage; thus, they cannot be applied for sound quantitative verification. In contrast, preimage *under-approximations* provide a lower bound on the volume of the preimage, allowing us to soundly verify quantitative properties.

5. Methodology

5.1 Overview

In this section, we present the main components of our methodology. Figure 3 shows the workflow of our preimage approximation method (using under-approximation as an illustration).

In Section 5.2, we introduce how to cheaply and soundly under-approximate (or over-approximate) the (restricted) preimage with a single polytope by means of the linear relaxation methods (Algorithm 1), which offer greater scalability than the exact method (Matoba and Fleuret, 2020). To handle the approximation loss caused by linear relaxation, in Section 5.3 we propose an anytime refinement algorithm that improves the approximation by partitioning a (sub)region into subregions with splitting (hyper)planes, with each subregion

Algorithm 1: GenApprox

```
Input: List of subregions $\mathcal{C}$, Output set $O$, Number of samples $N$, Boolean $Under$
Output: List of polytopes $\mathbf{T}$
$\mathbf{T} = []$;
for subregion $\mathcal{C}_{sub} \in \mathcal{C}$ // Parallel over subregions
do
    $x_1, ..., x_N \leftarrow \text{Sample}(\mathcal{C}_{sub}, N)$;
    if $Under$ then
        $[\underline{g_1}(x, \boldsymbol{\alpha}_1, \boldsymbol{\beta}_1), ..., \underline{g_K}(x, \boldsymbol{\alpha}_K, \boldsymbol{\beta}_K)] \leftarrow \text{LinearLowerBound}(\mathcal{C}_{sub}, O)$ ;
        $\text{Loss}(\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_K, \boldsymbol{\beta}_1, ..., \boldsymbol{\beta}_K) \leftarrow$
            $-\frac{\text{vol}(\mathcal{C}_{sub})}{N} \sum_{j=1,...,N} \sigma(-\text{LSE}(-\underline{g_1}(x_j, \boldsymbol{\alpha}_1, \boldsymbol{\beta}_1), ..., -\underline{g_K}(x_j, \boldsymbol{\alpha}_K, \boldsymbol{\beta}_K)))$;
        $\boldsymbol{\alpha}_1^*, ..., \boldsymbol{\alpha}_K^*, \boldsymbol{\beta}_1^*, ..., \boldsymbol{\beta}_K^* \leftarrow \text{argmin}(\text{Loss}(\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_K, \boldsymbol{\beta}_1, ..., \boldsymbol{\beta}_K))$;
        $\mathbf{T} = \text{Append}(\mathbf{T}, [\underline{g_1}(x, \boldsymbol{\alpha}_1^*, \boldsymbol{\beta}_1^*) \geq 0, ..., \underline{g_K}(x, \boldsymbol{\alpha}_K^*, \boldsymbol{\beta}_K^*) \geq 0, x \in \mathcal{C}_{sub}])$
    else
        $[\overline{g_1}(x, \boldsymbol{\alpha}_1, \boldsymbol{\beta}_1), ..., \overline{g_K}(x, \boldsymbol{\alpha}_K, \boldsymbol{\beta}_K)] \leftarrow \text{LinearUpperBound}(\mathcal{C}_{sub}, O)$ ;
        $\text{Loss}(\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_K, \boldsymbol{\beta}_1, ..., \boldsymbol{\beta}_K) \leftarrow$
            $\frac{\text{vol}(\mathcal{C}_{sub})}{N} \sum_{j=1,...,N} \sigma(-\text{LSE}(-\overline{g_1}(x_j, \boldsymbol{\alpha}_1, \boldsymbol{\beta}_1), ..., -\overline{g_K}(x_j, \boldsymbol{\alpha}_K, \boldsymbol{\beta}_K)))$;
        $\boldsymbol{\alpha}_1^*, ..., \boldsymbol{\alpha}_K^*, \boldsymbol{\beta}_1^*, ..., \boldsymbol{\beta}_K^* \leftarrow \text{argmin}(\text{Loss}(\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_K, \boldsymbol{\beta}_1, ..., \boldsymbol{\beta}_K))$;
        $\mathbf{T} = \text{Append}(\mathbf{T}, [\overline{g_1}(x, \boldsymbol{\alpha}_1^*, \boldsymbol{\beta}_1^*) \geq 0, ..., \overline{g_K}(x, \boldsymbol{\alpha}_K^*, \boldsymbol{\beta}_K^*) \geq 0, x \in \mathcal{C}_{sub}])$
return $\boldsymbol{T}$
```

then being approximated more accurately in parallel. In Section 5.4, we propose a novel differentiable objective to optimise the bounding parameters of linear relaxation to tighten the polytope approximation. Next, in Section 5.5, we propose a refinement scheme based on intermediate ReLU splitting planes and derive a preimage optimisation method using Lagrangian relaxation of the splitting constraints. The main contribution of this paper (Algorithm 2) integrates these four components and is described in Section 5.6. Finally, in Section 5.7, we apply our method to quantitative verification (Algorithm 3) and prove its soundness and completeness.

To simplify the presentation, we focus on computing under-approximations and explain the necessary changes to compute over-approximations in highlight boxes throughout.

5.2 Polytope Approximation via Linear Relaxation

We first show how to adapt linear relaxation techniques to efficiently generate valid under-approximations and over-approximations to the restricted preimage for a given input region $\mathcal{C}$ as a single polytope. Recall that LiRPA methods enable us to obtain linear lower and upper bounds on the output of a neural network f, that is, $\underline{\mathbf{A}}x + \underline{\mathbf{b}} \leq f(x) \leq \overline{\mathbf{A}}x + \overline{\mathbf{b}}$, where the linear coefficients depend on the input region $\mathcal{C}$.

Suppose that we are given the input hyperrectangle $\mathcal{C} = \{x \in \mathbb{R}^d | x \models \bigwedge_{i=1}^d \phi_i\}$, and the output polytope specified using the half-space constraints $\psi_i(y) = (c_i^T y + d_i \geq 0)$ for $i = 1, ..., K$ over the output space. Let us first consider generating a guaranteed under-approximation. Given a constraint ψ_i, we append an additional linear layer at the end of the network f, which maps $y \mapsto c_i^T y + d_i$, such that the function $g_i : \mathbb{R}^d \to \mathbb{R}$ represented by the new network is $g_i(x) = c_i^T f(x) + d_i$. Then, applying LiRPA lower bounding to each g_i, we obtain a lower bound $\underline{g_i}(x) = \underline{a}_i^T x + \underline{b}_i$ for each i, such that $\underline{g_i}(x) \geq 0 \implies g_i(x) \geq 0$ for $x \in \mathcal{C}$. Notice that, for each $i = 1, ..., K$, $\underline{a}_i^T x + \underline{b}_i \geq 0$ is a half-space constraint in the input space. We conjoin these constraints, along with the restriction to the input region $\mathcal{C}$, to obtain a polytope:

$$\underline{T_{\mathcal{C}}}(O) := \{x | \bigwedge_{i=1}^{K} (\underline{g_i}(x) \geq 0) \wedge \bigwedge_{i=1}^{d} \phi_i(x)\} \tag{2}$$

Over-Approximation Alternatively, to generate a guaranteed over-approximation, we can instead apply LiRPA upper bounding to each g_i, obtaining upper bounds $\overline{g_i}(x) = \overline{a}_i^T x + \overline{b}_i$ for each i, such that $g_i(x) \geq 0 \implies \overline{g_i}(x) \geq 0$ for $x \in \mathcal{C}$, and defining the polytope:

$$\overline{T_{\mathcal{C}}}(O) := \{x | \bigwedge_{i=1}^{K} (\overline{g_i}(x) \geq 0) \wedge \bigwedge_{i=1}^{d} \phi_i(x)\} \tag{3}$$

Proposition 6 *$\underline{T_{\mathcal{C}}}(O), \overline{T_{\mathcal{C}}}(O)$ are respectively under- and over-approximations to the restricted preimage $f_{\mathcal{C}}^{-1}(O)$.*

Proof For the under-approximation, the LiRPA bound $\underline{g_i}(x) \leq g_i(x)$ holds for any $x \in \mathcal{C}$ and $i = 1, ..., K$, and so we have $\bigwedge_{i=1}^{K} (\underline{g_i}(x) \geq 0) \wedge x \in \mathcal{C} \implies \bigwedge_{i=1}^{K} (g_i(x) \geq 0) \wedge x \in \mathcal{C}$, i.e., $\underline{T_{\mathcal{C}}}(O)$ is an under-approximation to $f_{\mathcal{C}}^{-1}(O)$. Similarly, for the over-approximation, $g_i(x) \leq \overline{g_i}(x)$ holds for any $x \in \mathcal{C}$ and $i = 1, ..., K$, and so $\bigwedge_{i=1}^{K} (g_i(x) \geq 0) \wedge x \in \mathcal{C} \implies \bigwedge_{i=1}^{K} (\overline{g_i}(x) \geq 0) \wedge x \in \mathcal{C}$, i.e. $\overline{T_{\mathcal{C}}}(O)$ is an over-approximation to $f_{\mathcal{C}}^{-1}(O)$.

■

Example 3 *Returning to Example 1, the output constraints (for $i = 2, 3, 4$) are given by $\psi_i = (y_1 - y_i \geq 0) = (c_i^T y + d_i \geq 0)$, where $c_i := e_1 - e_i$ (we use e_i to denote the i^{th} standard basis vector) and $d_i := 0$. Applying LiRPA bounding, we obtain the linear lower bounds $\underline{g_2}(x) = -4.12x_1 + x_2 + 2.98 \geq 0; \underline{g_3}(x) = 0.33x_1 - x_2 + 0.63 \geq 0$; and $\underline{g_4}(x)$ (not shown). The intersection of these constraints, shown in Figure 4 (region in grey), represents an under-approximation to the preimage. Similarly, we can obtain linear upper bounds $\overline{g_2}(x) = -12.23x_1 - x_2 + 15.18 \geq 0; \overline{g_3}(x) = -0.01x_1 - x_2 + 1.18 \geq 0$; and $\overline{g_4}(x) = -1.06x_1 - x_2 + 2.24 \geq 0$; the intersection of those constraints represents an over-approximation to the preimage, as shown in Figure 4 (region in blue).*

We generate the linear bounds in parallel over the output polyhedron constraints $i = 1, ..., K$ using the *backward mode* LiRPA (Zhang et al., 2018), and store the resulting approximating polytope as a list of constraints. This highly efficient procedure is used as a sub-routine `LinearBounds` when generating either preimage under-approximations or over-approximations in Algorithm 1 (Lines 6, 11).

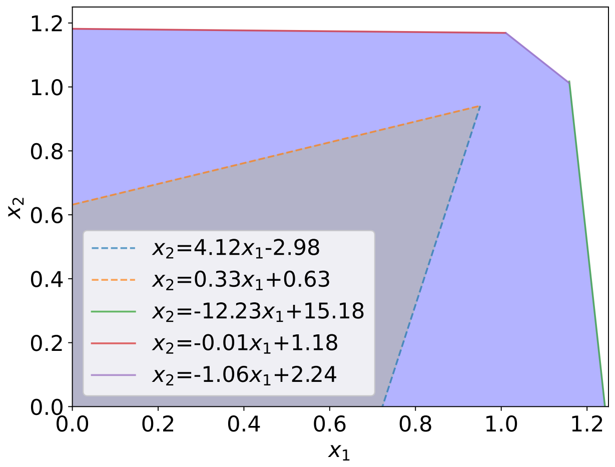


Figure 4: Illustration of initial preimage under- and over-approximation for output specification $\wedge_{i\in\{2,3,4\}}(y_1 - y_i \geq 0)$ in the vehicle parking task (for more details see Example 3). The under-approximation is the polytope in grey, bounded by dotted half-planes, and the over-approximation is the polytope in blue, bounded by solid half-planes. The ground-truth preimage for the output specification is the rectangular region $[0, 1] \times [0, 1]$.

Algorithm 2: Preimage Approximation

```
Input: Neural network $f$, Input region $\mathcal{C}$, Output region $O$, Volume threshold $v$, Maximum
       iterations $R$, Number of samples $N$, Boolean $Under$, Boolean $SplitOnInput$
Output: Disjoint union of polytopes $\mathcal{T}_{\text{Dom}}$
1  $T \leftarrow$ GenApprox($\mathcal{C}$, $O$, $N$, $Under$) ;                 // Initial preimage polytope
2  Dom $\leftarrow \{(\mathcal{C}, T, \text{CalcPriority}(T, Under))\}$ ;                   // Priority queue
   // $\mathcal{T}_{\text{Dom}}$ is the union of the under/over-approximating polytopes in Dom
3  while ((Under and EstimateVolume($\mathcal{T}_{\text{Dom}}$) $< v$) or
     ($\neg$Under and EstimateVolume($\mathcal{T}_{\text{Dom}}$) $> v$)) and Iterations $\leq R$ do
4      $\mathcal{C}_{\text{sub}}, T$, Priority $\leftarrow$ Pop(Dom) ;             // Subregion with highest priority
5      if $SplitOnInput$ then
6          $id \leftarrow$ SelectInputFeature(Feature$_I$, $Under$) ;   // Feature$_I$ is the set of input
             features/dimensions
7      else
8          $id \leftarrow$ SelectReLUNode(Node$_Z$, $Under$); // Node$_Z$ is the set of unstable ReLU
             nodes
9      $[\mathcal{C}^l_{sub}, \mathcal{C}^u_{sub}] \leftarrow$ SplitOnNode($\mathcal{C}_{sub}$, $id$);              // Split on the selected node
10     $T^l, T^u \leftarrow$ GenApprox($[\mathcal{C}^l_{sub}, \mathcal{C}^u_{sub}]$, $O$, $N$, $Under$) ;             // Generate preimage
11     Dom $\leftarrow$ Dom $\cup \{(\mathcal{C}^l_{sub}, T^l$, CalcPriority($T^l$, $Under$)),
         $(\mathcal{C}^u_{sub}, T^r$, CalcPriority($T^r$, $Under$))$\}$;                 // Disjoint polytope
12 return $\mathcal{T}_{\text{Dom}}$
```

5.3 Global Branching and Refinement

As LiRPA performs crude linear relaxation, the resulting bounds can be quite loose, even with optimisation over bounding parameters (as we will see in Section 5.4), meaning that the (single) polytope under-approximation or over-approximation is unlikely to be a good approximation to the preimage by itself. To address this challenge, we employ a divide-and-conquer approach that iteratively refines our approximation of the preimage. Starting from the initial region $\mathcal{C}$ at the root, our method generates a tree by iteratively partitioning a subregion $\mathcal{C}_{sub}$ represented at a leaf node into two smaller subregions $\mathcal{C}_{sub}^l, \mathcal{C}_{sub}^u$, which are then attached as children to that leaf node. In this way, the subregions represented by all leaves of the tree are disjoint, such that their union is the initial region $\mathcal{C}$.

In order to under-approximate (resp. over-approximate) the preimage, for each leaf subregion $\mathcal{C}_{sub}$ we compute, using LiRPA bounds, an associated polytope that under-approximates (resp. over-approximates) the preimage in $\mathcal{C}_{sub}$. Thus, irrespective of the number of refinements performed, the union of the under-approximating polytopes (resp. over-approximating) corresponding to all leaves forms an *anytime* DUP under-approximation (resp. over-approximation) $\mathcal{T}$ to the preimage in the original region $\mathcal{C}$. The process of refining the subregions continues until an appropriate termination criterion is met.

Unfortunately, even with a moderate number of input dimensions or unstable ReLU nodes, naïvely splitting along all input- or ReLU-planes quickly becomes computationally intractable. For example, splitting a d-dimensional hyperrectangle using bisections along each dimension results in 2^d subdomains to approximate. It thus becomes crucial to prioritise the subregions to split, as well as improve the efficiency of the splitting procedure itself. We describe these in turn.

Subregion Selection. We propose a subregion selection strategy that prioritises splitting subregions with the largest difference in volume between the exact preimage $f_{\mathcal{C}_{sub}}^{-1}(O)$ and the (already computed) polytope approximation $T_{\mathcal{C}_{sub}}(O)$ on that subdomain: this indicates "how much improvement" can be achieved on this subdomain and is implemented as the `CalcPriority` function in Algorithm 2. Unfortunately, computing the volume of a polytope exactly is a computationally expensive task, requiring specialised tools (Chevallier et al., 2022). To overcome this, we employ Monte Carlo estimation of volume computation by sampling N points $x_1, ..., x_N$ uniformly from the input subdomain $\mathcal{C}_{sub}$. For an under-approximation, we have:

$$\text{Priority}(\mathcal{C}_{sub}) := \frac{\text{vol}(\mathcal{C}_{sub})}{N} \times \left(\sum_{i=1}^{N} \mathbb{1}_{x_i \in f_{\mathcal{C}_{sub}}^{-1}(O)} - \sum_{i=1}^{N} \mathbb{1}_{x_i \in \underline{T_{\mathcal{C}_{sub}}}(O)} \right) \tag{4}$$

$$\approx \text{vol}(f_{\mathcal{C}_{sub}}^{-1}(O)) - \text{vol}(\underline{T_{\mathcal{C}_{sub}}}(O)) \tag{5}$$

This measures the gap between the polytope under-approximation and the optimal approximation, namely, the preimage itself.

Over-Approximation Similarly, in the case of an over-approximation, we define:

$$\text{Priority}(\mathcal{C}_{sub}) := \frac{\text{vol}(\mathcal{C}_{sub})}{N} \times \left(\sum_{i=1}^{N} \mathbb{1}_{x_i \in \overline{T_{\mathcal{C}_{sub}}}(O)} - \sum_{i=1}^{N} \mathbb{1}_{x_i \in f^{-1}_{\mathcal{C}_{sub}}(O)} \right) \tag{6}$$

$$\approx \text{vol}(\overline{T_{\mathcal{C}_{sub}}}(O)) - \text{vol}(f^{-1}_{\mathcal{C}_{sub}}(O)) \tag{7}$$

We then choose the leaf subdomain with the maximum priority. This leaf subdomain is then partitioned into two subregions $\mathcal{C}^l_{sub}, \mathcal{C}^u_{sub}$, each of which we then approximate with polytopes $T_{\mathcal{C}^l_{sub}}(O), T_{\mathcal{C}^u_{sub}}(O)$. Tighter intermediate concrete bounds, and thus tighter linear bounding functions, can often be computed on the partitioned subregions. While such locally improved bounds do not guarantee a global improvement in preimage volume, the polytope approximation on each subregion is typically refined compared to that on the original subregion. In our framework, we design a greedy input splitting strategy (see below) that selects the partition leading to the greatest improvement in preimage volume. Additionally, we perform optimisation (Section 5.4 and 5.5) over bounding parameters within each subregion to further enhance the tightness of the polytope approximation.

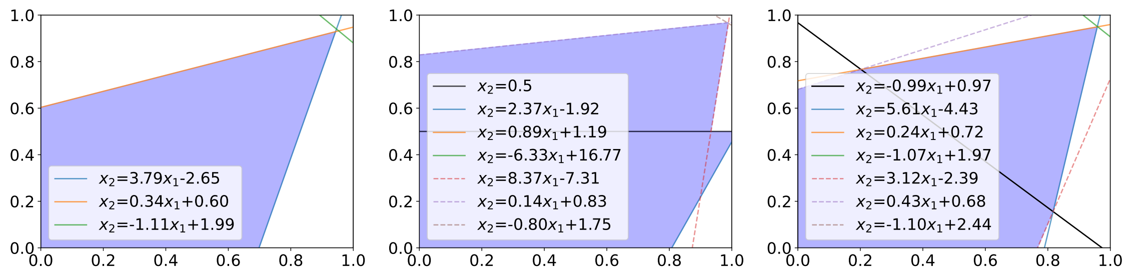
Notice that, although we approximate the volumes by sampling, this does not affect the *deterministic* volume guarantees provided by our method, as the Priority is simply a heuristic used to guide the algorithm. In the rest of this subsection, we consider how to split a leaf subregion into two subregions to optimise the volume of the preimage approximation. In particular, we propose two approaches: *input splitting* and *ReLU splitting.*

Input Splitting. Given a subregion (hyperrectangle) defined by lower and upper bounds $x_i \in [\underline{x}_i, \overline{x}_i]$ for all dimensions $i = 1, ..., d$, input splitting partitions it into two subregions by cutting along some feature i. This splitting procedure will produce two subregions that are similar to the original subregion, but have updated bounds $[\underline{x}_i, \frac{\underline{x}_i + \overline{x}_i}{2}], [\frac{\underline{x}_i + \overline{x}_i}{2}, \overline{x}_i]$ for feature i instead. A commonly-adopted splitting heuristic is to select the dimension with the longest edge (Bunel et al., 2020), that is, to select feature i with the largest range: $\arg\max_i(\overline{x}_i - \underline{x}_i)$. However, this method does not perform well in terms of per-iteration volume improvement of the preimage approximation.

Thus, we propose to greedily select a dimension instead according to a volume-aware heuristic. Specifically, for each feature, we generate approximating polytopes T^l, T^r for the two subregions resulting from the split, and choose the feature that maximises the following priority metric. In the case of under-approximation, when T^l consists of linear lower bounds $\underline{g_1}^l, \ldots, \underline{g_K}^l$ and T^r consists of linear lower bounds $\underline{g_1}^r, \ldots, \underline{g_K}^r$, we define:

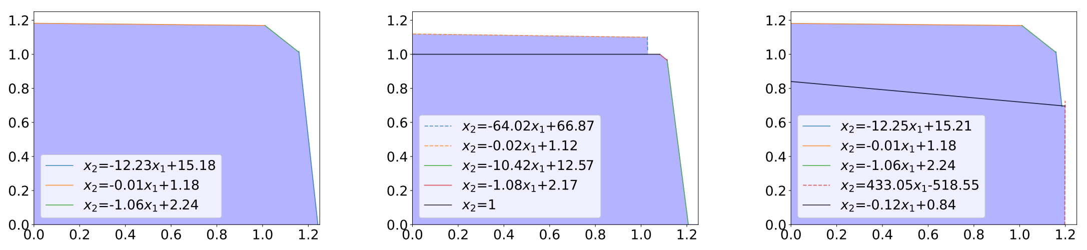
$$\text{InputPriority}(T^l, T^r) := \frac{\text{vol}(\mathcal{C}_{sub})}{N} \left(\sum_{j=1}^{N} \sigma\left(\min_{i=1,\ldots K} \underline{g_i^l}(x_j) \right) + \sum_{j=1}^{N} \sigma\left(\min_{i=1,\ldots K} \underline{g_i^r}(x_j) \right) \right) \tag{8}$$

where σ is the sigmoid function $\sigma(y) := \frac{1}{1+e^{-y}}$. Intuitively, this is an approximation to the (total) volume of the under-approximating polytopes (e.g., x_j is in the polytope T_l iff $\min_{i=1,\ldots K} \underline{g_i^l}(x_j) > 0$); we should prefer to split on input features that maximise the total volume. However, we found empirically that, in early iterations of the refinement, the



(a) Initial under-approximation (b) Input split (c) ReLU split

Figure 5: Refinement of the initial preimage under-approximation with input and ReLU splitting. Figure 5b and 5c display the refined preimage, i.e., larger volume, after adding input and ReLU splitting planes, where the dotted and solid bounding planes are used to form the polytope on each subregion, respectively.



(a) Initial over-approximation (b) Input split (c) ReLU split

Figure 6: Refinement of the initial preimage over-approximation with input and ReLU splitting. Figure 6b and 6c display the refined preimage, i.e., smaller volume, after adding input and ReLU splitting planes, where the dotted and solid bounding planes are used to form the polytope on each subregion, respectively.

under-approximation could often be empty (as the set $\{\min_{i=1,\ldots,K} \underline{g_i^l}(x_j) > 0\}$ for all i lies outside the subregion $\mathcal{C}_{sub}$), leading to zero priority for all features. For this reason, we propose to instead use the smooth sigmoid function to measure "how close" the constraints are to being satisfied for the sampled points, in order to provide signal for the best feature to split on.

Over-Approximation Similarly, if we are generating an over-approximation, then we prioritise according to the following (i.e., minimising volume):

$$\text{InputPriority}(T^l, T^r) := -\frac{\text{vol}(\mathcal{C}_{sub})}{N}\left(\sum_{j=1}^{N}\sigma\left(\min_{i=1,\ldots K}\overline{g_i^l}(x_j)\right) + \sum_{j=1}^{N}\sigma\left(\min_{i=1,\ldots K}\overline{g_i^r}(x_j)\right)\right) \tag{9}$$

Example 4 *We revisit Example 1. Figure 5a shows the initial polytope under-approximation computed on the input region $\mathcal{C}$ before refinement, where each solid line represents the bound-*

ing plane for each output specification ($y_1 - y_i \geq 0$). Figure 5b depicts the refined approximation by splitting the input region along the vertical axis, where the solid and dashed lines represent the bounding planes for the two resulting subregions. It can be seen that the total volume of the under-approximation has improved significantly. Similarly, in Figure 6a, we show the initial polytope over-approximation before refinement, and in Figure 6b the improved over-approximation after greedy input splitting.

Intermediate ReLU Splitting. Refinement through splitting on input features is adequate for low-dimensional input problems such as reinforcement learning agents. However, it may be infeasible to generate sufficiently fine subregions for high-dimensional domains. We thus propose an algorithm for ReLU neural networks that uses intermediate ReLU splitting for preimage refinement. After determining a subregion for refinement, we partition the subregion based upon the pre-activation value of an intermediate unstable neuron $z_j^{(i)} = 0$. As a result, the original subregion $\mathcal{C}_{sub}$ is split into two new subregions $\mathcal{C}^+_{z_j^{(i)}} = \{x \in \mathcal{C}_{sub} \mid z_j^{(i)} = h_j^{(i)}(x) \geq 0\}$ and $\mathcal{C}^-_{z_j^{(i)}} = \{x \in \mathcal{C}_{sub} \mid z_j^{(i)} = h_j^{(i)}(x) < 0\}$. Note that to obtain the polytope approximation, we can utilise linear lower/upper bounds on $h_j^{(i)}(x)$ as an approximation to the subregion boundary.

In this procedure, the order of splitting unstable ReLU neurons can greatly influence the quality and efficiency of the refinement. Existing heuristic methods for ReLU prioritisation select ReLU nodes that lead to greatest improvement in the final bound (maximum or minimum value) of the neural network f over the input domain (Bunel et al., 2020), e.g., improving $\min_{x\in\mathcal{C}} \underline{f}(x)$. These methods focus on optimising the worst-case output bounds at specific input points such as $x^* = \arg\min_{x\in\mathcal{C}} \underline{f}(x)$. However, these methods are not well-suited for preimage analysis, where our aim is instead to refine the preimage approximation to the exact preimage. Specifically, the objective is to minimise the volume of under-approximations and maximise volumes of over-approximations. Prioritising ReLU nodes based on local worst-case bound improvements may therefore fail to improve the overall precision of the preimage approximation. To this end, we compute (an estimate of) the volume difference between the split subregions $|\mathrm{vol}(\mathcal{C}^+_{z_j^{(i)}}) - \mathrm{vol}(\mathcal{C}^-_{z_j^{(i)}})|$, using a single forward pass for a set of sampled data points from the input domain; note that this is bounded above by the total subregion volume $\mathrm{vol}(\mathcal{C}_{sub})$. We then propose to select the ReLU node that minimises this difference. Intuitively, this choice results in balanced subdomains after splitting.

A key advantage of ReLU splitting is that it allows us to replace unstable neuron bounds with precise bounds. For an unstable ReLU neuron $a_j^{(i)}(x) = \max(0, h_j^{(i)}(x))$, we use linear relaxation to bound the post-activation value (as in Equation 1), which yields linear lower and upper bounds of the form $\underline{c}h_j^{(i)}(x) + \underline{d} \leq a_j^{(i)}(x) \leq \overline{c}h_j^{(i)}(x) + \overline{d}$, where $\underline{c}$, $\overline{c}$ denote the slopes and $\underline{d}$, $\overline{d}$ denote the intercepts of the lower and upper bounding functions, respectively. When a split is introduced, the neuron becomes stable in each subdomain, and the exact linear function $a_j^{(i)}(x) = h_j^{(i)}(x)$ and $a_j^{(i)}(x) = 0$ can be used in place of its linear relaxation, as shown in Figure 1 (unstable to stable). This can typically tighten the approximation on each subdomain as the linear relaxation errors for this unstable neuron are removed for each subdomain and substituted with the exact symbolic function for backward propagation.

Example 5 *We now apply our algorithm with ReLU splitting to the problem in Example 1. Figure 5c shows the refined preimage polytope by adding the splitting plane (black solid line) along the direction of a selected unstable ReLU node. Compared with Figure 5a, we can see that the volume of the approximation increased. Similarly, in Figure 6c, we show the improved over-approximation after ReLU splitting, compared to the initial over-approximation 6a.*

Combining Preimage Polytopes. As the final step, we combine the refined symbolic approximations on each subregion to compute the disjoint polytope union for the desired preimage of the output property. Note that the input splitting (hyper)planes naturally yield disjoint subregions. We can directly compute the final disjoint polytope union by combining the preimage polytopes of each subregion, where the splitting planes serve as part of the constraints that form the preimage polytope, e.g., two disjoint polytopes with the splitting constraints $x_2 - 0.5 \geq 0$ and $-x_2 + 0.5 \geq 0$, respectively, partitioned by $x_2 = 0.5$ in Figure 5b. In the case of ReLU splitting, as each ReLU neuron represents a complex non-linear function with respect to the input, we cannot directly add the constraints introduced by ReLU splitting to the polytope representation. Instead, we compute the linear upper or lower bounding functions of the non-linear constraint represented by the ReLU neuron, i.e., $\underline{h_j^{(i)}}(x) \leq h_j^{(i)}(x) \leq \overline{h_j^{(i)}}(x)$. The constraints introduced by the linear bounding functions, i.e., $\underline{h_j^{(i)}}(x) \geq 0$ and $-\underline{h_j^{(i)}}(x) \geq 0$, can then be added to form disjoint polytopes. For instance, as shown in Figure 5c, two disjoint polytopes are formed with the additional splitting constraints $-0.99x_1 - x_2 + 0.97 \geq 0$ and $0.99x_1 + x_2 - 0.97 \geq 0$, respectively, partitioned by the linear splitting plane $x_2 = -0.99x_1 + 0.97$ (exact linear function of the selected ReLU neuron in this case). In fact, any linear function between the linear upper and lower bounding functions of the ReLU neuron serves as a valid splitting (hyper)plane to form disjoint polytopes.

Input vs ReLU Splitting. Input and ReLU splitting are alternative strategies for splitting into smaller subregious that users can employ for different application scenarios, but not both simultaneously (Lines 4-8 in Algorithm 2). The choice of which to use should primarily be guided by the dimensionality of the problem; we found in our experiments that input splitting is quite effective for low-dimensional problems (see Section 6.2.5), while ReLU splitting is necessary to scale to high dimensions. This is consistent with findings from LiRPA-based verification tools such as α-β-CROWN (Wang et al., 2021b) and our comparison experiments in Section 6.2.5.

5.4 Local Optimisation

One of the key components behind the effectiveness of LiRPA-based bounds is the ability to efficiently improve the tightness of the bounding function by optimising the relaxation parameters $\boldsymbol{\alpha}$ via projected gradient descent. In the context of local robustness verification, the goal is to optimise the *concrete* (scalar) lower or upper bounds over the (sub)region $\mathcal{C}_{sub}$ (Xu et al., 2020), i.e., $\min_{x \in \mathcal{C}_{sub}} \underline{\mathbf{A}}(\boldsymbol{\alpha})x + \underline{\mathbf{b}}(\boldsymbol{\alpha})$ in the case of lower bounds, where we explicitly note the dependence of the linear coefficients on $\boldsymbol{\alpha}$. In our case, we are instead interested in optimising $\boldsymbol{\alpha}$ to refine the polytope approximation, that is, increase the volume of under-approximations and decrease the volume of over-approximations (to the exact preimage).

As before, we employ statistical estimation; we sample N points $x_1, ..., x_N$ uniformly from the input domain $\mathcal{C}_{sub}$ then employ Monte Carlo estimation for the volume of the approximating polytope. In the case of under-approximation, we have:

$$\widehat{\text{vol}}(\underline{T_{\mathcal{C}_{sub},\boldsymbol{\alpha}}}(O)) = \frac{\text{vol}(\mathcal{C}_{sub})}{N} \times \sum_{i=1}^{N} \mathbb{1}_{x_i \in \underline{T_{\mathcal{C}_{sub},\boldsymbol{\alpha}}}(O)} \tag{10}$$

where we highlight the dependence of $\underline{T_{\mathcal{C}_{sub}}}(O) = \{x | \bigwedge_{i=1}^{K} \underline{g_i}(x, \boldsymbol{\alpha}_i) \geq 0 \wedge \bigwedge_{i=1}^{d} \phi_i(x)\}$ on $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_K)$, and $\boldsymbol{\alpha}_i$ are the α-parameters for the linear relaxation of the neural network g_i corresponding to the i^{th} half-space constraint in O. However, this is still non-differentiable w.r.t. $\boldsymbol{\alpha}$ due to the indicator function. We now show how to derive a differentiable relaxation, which is amenable to gradient-based optimization:

$$\widehat{\text{vol}}(\underline{T_{\mathcal{C}_{sub},\boldsymbol{\alpha}}}(O)) = \frac{\text{vol}(\mathcal{C}_{sub})}{N} \sum_{j=1}^{N} \mathbb{1}_{x_j \in \underline{T_{\mathcal{C}_{sub},\boldsymbol{\alpha}}}(O)} = \frac{\text{vol}(\mathcal{C}_{sub})}{N} \sum_{j=1}^{N} \mathbb{1}_{\min_{i=1,...K} \underline{g_i}(x_j, \boldsymbol{\alpha}_i) \geq 0} \tag{11}$$

$$\approx \frac{\text{vol}(\mathcal{C}_{sub})}{N} \sum_{j=1}^{N} \sigma \left(\min_{i=1,...K} \underline{g_i}(x_j, \boldsymbol{\alpha}_i) \right) \tag{12}$$

$$\approx \frac{\text{vol}(\mathcal{C}_{sub})}{N} \sum_{j=1}^{N} \sigma \left(-\text{LSE}(-\underline{g_1}(x_j, \boldsymbol{\alpha}_1), ..., -\underline{g_K}(x_j, \boldsymbol{\alpha}_K)) \right) \tag{13}$$

As before, we use a sigmoid relaxation to approximate the volume. However, the minimum function is still non-differentiable. Thus, we approximate the minimum over specifications using the log-sum-exp (LSE) function. The log-sum-exp function is defined by $LSE(y_1, ..., y_K) := \log(\sum_{i=1,...,K} e^{y_i})$, and is a differentiable approximation to the maximum function; we employ it to approximate the minimisation by adding the appropriate sign changes. The final expression is now a differentiable function of $\boldsymbol{\alpha}$.

Then the goal is to maximise the volume of the under-approximation with respect to $\boldsymbol{\alpha}$:

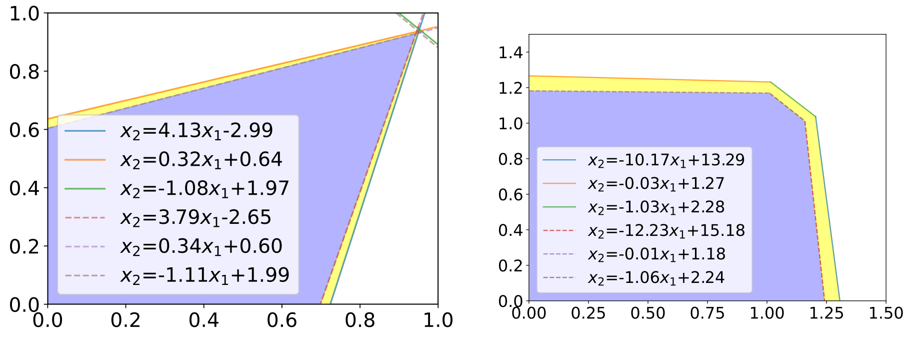
$$\text{Loss}(\boldsymbol{\alpha}) = -\widehat{\text{vol}}(\underline{T_{\mathcal{C}_{sub},\boldsymbol{\alpha}}}(O)) \tag{14}$$

We employ this as the loss function in Algorithm 1 (Line 7) for generating a polytope approximation, and optimise volume using projected gradient descent.

Over-Approximation In the case of an over-approximation (Line 12 of Algorithm 1), we instead aim to minimise the volume of the approximation:

$$\text{Loss}(\boldsymbol{\alpha}) = \widehat{\text{vol}}(\overline{T_{\mathcal{C}_{sub},\boldsymbol{\alpha}}}(O)) \tag{15}$$

Example 6 *We revisit Example 1. Figure 7a shows the computed under-approximations before and after local optimisation. We can see that the bounding planes for all three specifications are optimised, such that the volume of the approximation has increased. Similarly,*



(a) Optimised under-approximation (b) Optimised over-approximation

Figure 7: Local optimisation for preimage under- and over-approximations. In Figure 7a, the blue polytope represents the preimage under-approximation before optimisation and the yellow region illustrates the expanded polytope volume after optimisation. In Figure 7b, the blue polytope represents the over-approximation before optimisation. The yellow region shows the reduced polytope volume after optimisation.

in Figure 7b we show the over-approximations before and after optimisation; it can be seen that the volume of the over-approximation has decreased.

5.5 Optimisation of Lagrangian Relaxation

Previously, in Section 5.3, we proposed a preimage refinement method that adds intermediate ReLU splitting planes to tighten the bounds of a selected individual neuron. However, intermediate bounds for other neurons are not updated based on the newly added splitting constraint. In the following, we first discuss the impact of stabilising an intermediate ReLU neuron from two different perspectives. We then present an optimisation approach leveraging Lagrangian relaxation to enforce the splitting constraint on refining the preimage.

Effect of Stabilisation of Intermediate Neurons. Our previous approach of Zhang et al. (2024b) exploits one level of bound tightening after ReLU splitting: the substitution of relaxation functions with exact linear functions for the individual neuron. Specifically, assume an intermediate (unstable) neuron $z_j^{(i)}$ ($= h_j^{(i)}(x)$) is selected to split the input (sub)region $\mathcal{C}$ into two subregions $\mathcal{C}^+_{z_j^{(i)}} = \{x \in \mathcal{C} \mid z_j^{(i)} \geq 0\}$ and $\mathcal{C}^-_{z_j^{(i)}} = \{x \in \mathcal{C} \mid z_j^{(i)} < 0\}$. For each subregion, the linear bounding functions of the nonlinear activation function $a_j^{(i)}(z_j^{(i)})$, as shown in Figure 1 (unstable mode), are then substituted with the exact ones, eliminating relaxation errors on the particular neuron. Another effect, potentially more impactful, is the bound tightening of every other intermediate neuron. Intuitively, one can tighten the intermediate bounds of (and thus stabilise) the other unstable neurons, since we are restricted to a smaller input region with the added splitting plane. A straightforward solution to enforce the effect of the splitting constraint is to call a regular LP solver to compute the new lower and upper bounds for every intermediate ReLU neuron under the splitting constraints. Naturally, this is computationally expensive ($2N$ LP calls where N is the number of ReLU neurons).

Refinement with Optimisation of Lagrangian Relaxation. In order to derive tighter preimage approximations without explicitly introducing LP solver calls, we propose to adapt Lagrangian optimisation techniques (Wang et al., 2021b) to preimage generation.

Consider first the case of generating under-approximations. Without loss of generality, we focus on preimage generation for the k-th output specification constraint, $\underline{g_k}(x) = \underline{a}_k^T x + \underline{b}_k$. We will drop the subscript k for simplicity.

Consider the subregion where we have $z_j^{(i)} \leq 0$. To tighten the bounding plane $\underline{g}(x)$ of the preimage under the splitting constraint $z_j^{(i)} \leq 0$, we introduce the Lagrange multiplier, parameterized as $\beta_j^{(i)} (\geq 0)$, to enforce its effect throughout the neural network. Specifically, let us write $g(\boldsymbol{z}^{(i)})$ to denote the neural network function mapping from the pre-activation neurons $\boldsymbol{z}^{(i)}$ of layer i to the output. In the typical backward LiRPA propagation through layer i, we have:

$$g(\boldsymbol{z}^{(i)}) \geq \underline{\mathbf{A}}^{(i)} \boldsymbol{z}^{(i)} + \underline{\mathbf{b}}^{(i)} \tag{16}$$

Now, we add the splitting constraint using a Lagrange multiplier and obtain a Lagrangian relaxation of the original problem as follows:

$$g(\boldsymbol{z}^{(i)}) \geq \max_{\beta_j^{(i)} \geq 0} \underline{\mathbf{A}}^{(i)} \boldsymbol{z}^{(i)} + \underline{\mathbf{b}}^{(i)} + \beta_j^{(i)} z_j^{(i)} \tag{17}$$

Note that $\max_{\beta_j^{(i)} \geq 0} \beta_j^{(i)} z_j^{(i)} = 0$, and thus Equation 17 holds in the universally quantified region. For the other case where $z_j^{(i)} \geq 0$, we can obtain a sound lower bound similarly by changing the sign for the additional splitting constraint:

$$g(\boldsymbol{z}^{(i)}) \geq \max_{\beta_j^{(i)} \geq 0} \underline{\mathbf{A}}^{(i)} \boldsymbol{z}^{(i)} + \underline{\mathbf{b}}^{(i)} - \beta_j^{(i)} z_j^{(i)} \tag{18}$$

We then propagate this backwards through the network to obtain a valid lower bound with respect to the input layer x:

$$g(x) \geq \underline{g}(x) = \max_{\boldsymbol{\beta} \geq 0} \underline{\mathbf{A}}(\boldsymbol{\alpha}, \boldsymbol{\beta}) x + \underline{\mathbf{b}}(\boldsymbol{\alpha}, \boldsymbol{\beta}) \tag{19}$$

Here, we explicitly note the dependence of the linear coefficients on $\boldsymbol{\beta}$, which denotes the vector of $\beta_j^{(i)}$ introduced for all split neurons. These coefficients can be computed using a single (modified) LiRPA backward pass, where we add the Lagrange multipliers in each step as in Equations 17, 18. Once we obtain the bounding plane for each half-space constraint in O, the preimage polytope can be formulated as $T_{\mathcal{C}}(O) = \{x | \bigwedge_{i=1}^{K} \underline{g_i}(x, \boldsymbol{\alpha}_i, \boldsymbol{\beta}_i) \geq 0 \wedge \bigwedge_{i=1}^{d} \phi_i(x)\}$.

Similarly to the optimisation over relaxation parameters $\boldsymbol{\alpha}$, we can then optimise $\boldsymbol{\beta}$ to maximise the preimage volume. Our differentiable preimage volume estimate is given by:

$$\widehat{\mathrm{vol}}(T_{\mathcal{C}_{sub}, \boldsymbol{\alpha}, \boldsymbol{\beta}}(O)) = \frac{\mathrm{vol}(\mathcal{C})}{N} \sum_{j=1}^{N} \sigma \left(-\mathrm{LSE}(-\underline{g_1}(x_j, \boldsymbol{\alpha}_1, \boldsymbol{\beta}_1), ..., -\underline{g_K}(x_j, \boldsymbol{\alpha}_K, \boldsymbol{\beta}_K)) \right) \tag{20}$$

where we have added the dependence on the Lagrange multipliers $\boldsymbol{\beta}$ to Equation 13. Intuitively, the additional splitting constraint enforced by the Lagrangian relaxation reduces the

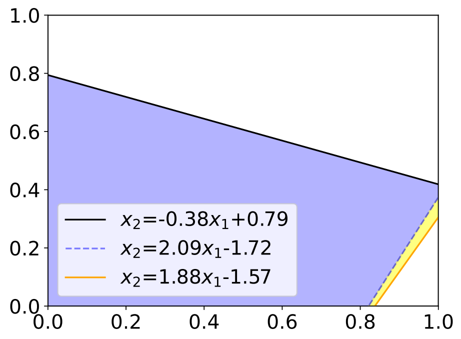


Figure 8: Optimisation of Lagrangian relaxation for preimage refinement. The preimage polytope in blue represents the under-approximation before Lagrangian optimisation and the yellow region displays the expanded polytope after optimisation. Details in Example 7.

input space for maximising the preimage volume, which allows a tighter preimage bounding plane for the subregion. In the case where all $\boldsymbol{\beta}$ coefficients are zero, this corresponds precisely to the previous standard LiRPA bound with $\boldsymbol{\alpha}$ parameters from Section 5.4. We then maximise the volume estimate of the under-approximation with the following loss function in Algorithm 1 (Line 7):

$$\text{Loss}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = -\widehat{\text{vol}}(\underline{T_{\mathcal{C}_{sub}, \boldsymbol{\alpha}, \boldsymbol{\beta}}}(O)) \tag{21}$$

Over-Approximation In the case of an over-approximation (Line 12 of Algorithm 1), we instead aim to minimise the volume of the approximation:

$$\text{Loss}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \widehat{\text{vol}}(\overline{T_{\mathcal{C}_{sub}, \boldsymbol{\alpha}, \boldsymbol{\beta}}}(O)) \tag{22}$$

Example 7 *We now apply our optimisation method over Lagrangian relaxation to Example 1. Figure 8 shows the preimage polytope before and after Lagrangian optimisation, respectively, where the splitting plane of the selected unstable ReLU node is marked with a black solid line. Note that the blue polytope before Lagrangian optimisation in Figure 8 is computed by removing the relaxation errors of the selected unstable ReLU node, where the symbolic upper/lower bounding functions are substituted with the exact linear functions. The preimage is further refined by enforcing the added splitting constraint $z_j^{(i)} \leq 0$ for one subdomain throughout the neuron network, which allows tighter preimage approximation (vs the blue polytope tightened via stabilizing a single neuron).*

5.6 Overall Algorithm

Our overall preimage approximation method is summarised in Algorithm 2. It takes as input a neural network f, input region $\mathcal{C}$, output region O, target polytope volume threshold v (a proxy for approximation precision), maximum number of iterations R, number of samples N for statistical estimation, and Boolean variables indicating (i) whether to return an under-approximation or over-approximation and (ii) whether to use input or ReLU splitting, and

```
Algorithm 3: Quantitative Verification
   Input: Neural network $f$, Property $(I, O, p)$, Maximum iterations $R$
   Output: Verification result $\in$ {True, False, Unknown}
1  $\text{vol}(I) \leftarrow \text{EstimateVolume}(I)$;
2  $\mathcal{C} \leftarrow \text{OuterBox}(I)$ ;                 // For general polytopes $I$
3  $\mathcal{T} \leftarrow \text{InitialRun}(f, \mathcal{C}, O)$;
4  while Iterations $\leq R$ do
5      $\mathcal{T} \leftarrow \text{Refine}(f, \mathcal{T}, \mathcal{C}, O)$;
6      if $\text{EstimateVolume}(\mathcal{T}) \geq p \times \text{vol}(I)$ then
7          return True
8      if AllReLUSplit then
9          return False
10 return Unknown
```

returns a disjoint polytope union $\mathcal{T}_{\text{Dom}}$ representing a guaranteed under-approximation (or over-approximation) to the preimage.

The algorithm initiates and maintains a priority queue of (sub)regions according to Equation 8. The *initialisation* step (Lines 1-2) generates an initial polytope approximation of the whole region using Algorithm 1 (Sections 5.2, 5.4, 5.5), with priority calculated (`CalcPriority`) according to Equations 5, 7. Then, the *preimage refinement* loop (Lines 3-11) partitions a subregion in each iteration, with the preimage restricted to the child subregions then being re-approximated (Line 9-10). In each iteration, we choose the region to split (Line 4) and the splitting plane to cut on (Line 6 for input split and Line 8 for ReLU split), as detailed in Section 5.3. The preimage subregion queue is then updated by computing the priorities for each subregion by approximating their volume (Line 11). The loop terminates and the approximation is returned when the target volume threshold v or maximum iteration limit R is reached.

5.7 Quantitative Verification

We now show how to use our efficient preimage under-approximation method (Algorithm 2) to verify a given quantitative property (I, O, p), where O is a polyhedron, I a polytope and p the desired proportion threshold, summarised in Algorithm 3. Note that preimage over-approximation cannot be applied for sound quantitative verification as the approximation may contain false regions outside the true preimage. To simplify, assume that I is a hyperrectangle, so that we can take $\mathcal{C} = I$. We discuss the case of general polytopes at the end of this section.

We utilise Algorithm 2 by setting the volume threshold v to $p \times \text{vol}(I)$, such that we have $\frac{\text{vol}(\mathcal{T})}{\text{vol}(I)} \geq p$ if the algorithm terminates before reaching the maximum number of iterations. If the final preimage polytope volume $\text{vol}(\mathcal{T}) \geq p \times \text{vol}(I)$, then the property is verified. Otherwise, we continue running the preimage refinement. If the refinement loop has stabilised all ReLU neurons and the volume threshold is still not achieved, the property is falsified.

In Algorithm 3, `InitialRun` generates an initial under-approximation to the preimage as in Lines 1-2 of Algorithm 1, and `Refine` performs one iteration of approximation refinement (Lines 4-11). Termination occurs when we have verified or falsified the quantitative property, or when the maximum number of iterations has been exceeded.

Proposition 7 *Algorithm 3 is sound for quantitative verification with input splitting.*

Proposition 8 *Algorithm 3 is sound and complete for quantitative verification on piecewise linear neural networks with ReLU splitting.*

Proofs of the propositions are presented in Appendix B.

General Input Polytopes. Previously we detailed how to use our preimage under-approximation method to verify quantitative properties (I, O, p), where I is a hyperrectangle. We now discuss how to extend our method for a general polytope $I = \{x \in \mathbb{R}^d | \bigwedge_{i=1}^{K_{in}} c_i^T x + d_i \geq 0\}$, (where ψ_i are half-space constraints). Intuitively, we can handle these general input polytopes (i) by finding a bounding hyperrectangle $\mathcal{C}$; and (ii) by encoding the polytope half-planes as additional constraints defining the preimage. Assuming that the output polytope O is given as $g_i(x) \geq 0$ for $i = 1, \ldots, K_{out}$, then we have:

$$f_{\mathcal{C}}^{-1}(O) \cap I := \left\{ x \in \mathbb{R}^d \middle| \bigwedge_{i=1}^{K_{out}} g_i(x) \geq 0 \wedge x \in \mathcal{C} \wedge \bigwedge_{i=1}^{K_{in}} c_i^T x + d_i \geq 0 \right\} \tag{23}$$

Firstly, in Line 2 of Algorithm 3, we derive a hyperrectangle $\mathcal{C}$ such that $I \subseteq \mathcal{C}$, by converting the polytope I into its *V-representation* (Grünbaum et al., 2003), that is, a list of the vertices (extreme points) of the polytope, which can be computed as in Avis and Fukuda (1991); Barber et al. (1996). Once we have a V-representation, obtaining a bounding box (hyperrectangle) can be achieved simply by computing the minimum and maximum value $\underline{x_i}, \overline{x_i}$ of each dimension among all vertices.

Once we have this input hyperrectangle $\mathcal{C}$, we can then run the preimage refinement as usual, but with the modification that, when defining the polytopes and restricted preimages, we must additionally include the polytope constraints from I. Practically, this means that, during every call to `EstimateVolume` in Algorithm 3, we add these polytope constraints, and in Line 9 of Algorithm 1 we add the polytope constraints from I, in addition to those derived from the output O and the box constraints from $\mathcal{C}_{sub}$. The output will be a DUP under/over-approximation of the preimage intersected with the polytope I.

6. Experiments

We have implemented our approach as a tool (Zhang et al., 2025) for preimage approximation for polyhedral output sets/specifications. In this section, we report on experimental evaluation of the proposed approach, and demonstrate its effectiveness in approximation generation and the application to quantitative analysis of neural networks.

Table 1: Performance comparison on preimage generation, for four different specifications on the vehicle parking task. Over-approximation results are highlighted with *grey* background in subcolumns labelled by **ox**, whereas under-approximation is shown with *white* background in subcolumns labelled by **ux**.

Vehicle Parking	**Exact**		**Invprop**				**PREMAP**					
	#Poly	**Time(s)**	**Time(s)**		**Cov**		**#Poly**		**Time(s)**		**Cov**	
	exact	**exact**	**ux**	**ox**	**ux**	**ox**	**ux**	**ox**	**ux**	**ox**	**ux**	**ox**
$\wedge_{i\in\{2,3,4\}} y_1 \geq y_i$	10	3110.979	2.642	**0.907**	0.921	**1.043**	**4**	**4**	**1.116**	1.121	**0.957**	1.092
$\wedge_{i\in\{1,3,4\}} y_2 \geq y_i$	20	3196.561	2.242	**0.793**	0.895	**1.051**	**4**	**4**	**1.235**	1.336	**0.948**	1.074
$\wedge_{i\in\{1,2,4\}} y_3 \geq y_i$	7	3184.298	2.325	**0.865**	0.906	**1.083**	**3**	**4**	**1.074**	1.129	**0.952**	1.098
$\wedge_{i\in\{1,2,3\}} y_4 \geq y_i$	15	3206.998	2.402	**0.793**	0.915	**1.058**	**3**	**3**	**1.055**	1.004	**0.922**	1.061

6.1 Benchmark and Evaluation Metric

We evaluate PREMAP on a benchmark of reinforcement learning and image classification tasks. Besides the vehicle parking task of Ayala et al. (2011) shown in the running example, we consider the following tasks: (1) aircraft collision avoidance system (VCAS) from Julian and Kochenderfer (2019) with 9 feed-forward neural networks (FNNs); (2) neural network controllers from Müller et al. (2022) for three reinforcement learning tasks (Cartpole, Lunarlander, and Dubinsrejoin) as in Brockman et al. (2016); and (3) the neural network for MNIST classification from VNN-COMP 2022 (Brix and Shi, 2022). Details of the benchmark tasks and neural networks are summarised in Appendix A.

Evaluation Metric. To evaluate the quality of the preimage approximation, we define the *coverage ratio* to be the ratio of volume covered by the approximation to the volume of the exact preimage, i.e., $\text{cov}(\mathcal{T}, f_{\mathcal{C}}^{-1}(O)) := \frac{\text{vol}(\mathcal{T})}{\text{vol}(f_{\mathcal{C}}^{-1}(O))}$. Note that this is a normalised measure for assessing the quality of the approximation, as used in Algorithm 3 when comparing with target coverage proportion p for termination of the refinement loop. In practice, we use Monte Carlo estimation to compute $\text{vol}(f_{\mathcal{C}}^{-1}(O))$ as $\widehat{\text{vol}}(f_{\mathcal{C}}^{-1}(O)) = \text{vol}(\mathcal{C}) \times \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{f(x_i)\in O}$, where $x_1, ...x_N$ are samples from $\mathcal{C}$. In Algorithm 2, the target volume (stopping criterion) is set as $v = r \times \widehat{\text{vol}}(f_{\mathcal{C}}^{-1}(O)$, where r is the *target coverage ratio*. Thus, in our experimental results, the coverage ratio reported will be greater than (resp. less than) the target coverage ratio for an under-approximation (resp. over-approximation), unless the maximum number of iterations is reached.

6.2 Evaluation

6.2.1 Effectiveness on Preimage Approximation with Input Split

We apply Algorithm 2 with input splitting to the preimage approximation problem for low-dimensional reinforcement learning tasks. For comparison, we also run the exact preimage generation method (Exact) from Matoba and Fleuret (2020) and the preimage over-approximation method (Invprop) from Kotha et al. (2023, accessed October, 2023).

Vehicle Parking & VCAS. Table 1 and 2 present the comparison results with state-of-the-art exact and approximate preimage generation methods. In the table, we show the number of polytopes (#Poly) in the preimage, computation time (Time(s)), and the approximate coverage ratio (Cov) when the preimage approximation algorithm terminates

Table 2: Performance comparison on preimage generation (average performance) on vehicle parking and VCAS, with over-approximation shown in *grey* background (subcolumns labelled by **ox**) and under-approximation in *white* background (subcolumns labelled by **ux**).

Tasks	**Exact**		**Invprop**				**PREMAP**					
	#Poly	**Time(s)**	**Time(s)**		**Cov**		**#Poly**		**Time(s)**		**Cov**	
	exact	**exact**	**ux**	**ox**	**ux**	**ox**	**ux**	**ox**	**ux**	**ox**	**ux**	**ox**
Vehicle	13	3174.709	2.403	**0.840**	0.909	**1.059**	**4**	**4**	**1.120**	1.148	**0.945**	1.081
VCAS	131	6363.272	-	-	-	-	**15**	**1**	**10.775**	**1.045**	**0.908**	**1.041**

with target coverage of 0.90 (the larger, the better) for under-approximation and 1.10 (the lower, the better) for over-approximation. Note that the *Exact* method computes the exact preimage (i.e., coverage ratio 1.0), while PREMAP computes the under- and over-approximation of the exact preimage. The results for over-approximation are highlighted with *grey* background, whereas under-approximation is shown with *white* background. Invprop only supports computing over-approximations natively; thus, we adapt it to produce an under-approximation by computing over-approximations for the complement of each output constraint; note that the resulting approximation is then the *complement* of a union of polytopes, rather than a DUP.

Compared with the exact method, our approach yields *orders-of-magnitude* improvement in efficiency (see Table 1 and Table 2). It can also characterise the preimage with much fewer (and also disjoint) polytopes, achieving an average reduction of 69.2% for vehicle parking (both under- and over-approximation) and 88.5% (under-approximation) and 99.2% (over-approximation) for VCAS. Compared with Invprop, our method produces comparable results in terms of time and approximation coverage on the 2D vehicle parking task. In Table 2, Invprop provides a tighter over-approximation and lower runtime for the vehicle parking task. This advantage can be attributed to Invprop's exploitation of output constraints for iterative refinement of intermediate bounds, which improves precision and reduces the number of splitting rounds needed. However, the iterative refinement procedure utilising output constraints in Invprop is quite expensive. It essentially has a quadratic dependence on network depth when all intermediate bounds are refined because refining the bounds of each intermediate layer requires bound propagation to the input layer. In contrast, our framework utilises output constraints with a linear dependence with respect to the number of layers. As a result, Invprop's scalability to deeper neural networks remains a limitation, especially in more complex tasks or architectures with greater depth.

While Table 2 shows average performance on VCAS, Figure 9 plots more detailed results of our method for the nine neural networks in the VCAS task in terms of the number of polytopes (y-axis on the left), time cost (y-axis on the left) and approximation coverage (y-axis on the right) for both under- (indicated with *metric*_U) and over-approximation (indicated with *metric*_O). As shown in the figure, our method is able to reach the targeted approximation coverage (0.90 for under-approximation and 1.10 for over-approximation) for all networks. The median number of polytopes for the preimage under-approximation for property $O = \{y \in \mathbb{R}^9 \mid \wedge_{i \in [1,9]} \;\; y_1 \geq y_i\}$ is 15 and the median time cost is 10.492s. The over-approximation shows higher variability in the number of generated polytopes and

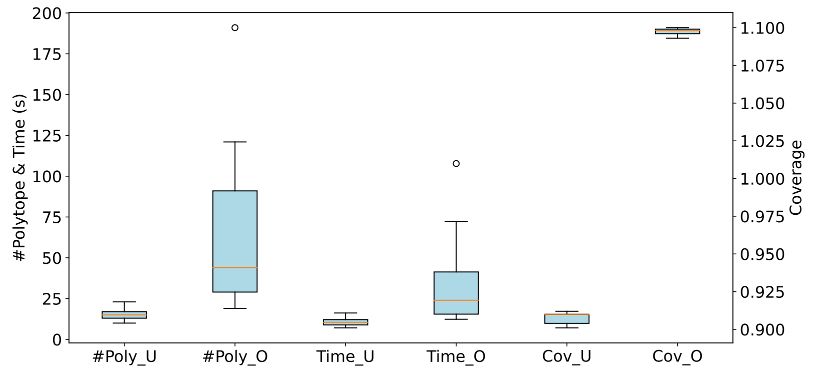


Figure 9: Preimage approximation results over the 9 VCAS neural networks. Results for under-approximation are indicated with *metric*_U and over-approximation with *metric*_O. The scale for both the number of polytopes and time is indicated on the vertical axis on the left.

Table 3: Performance of preimage approximation for reinforcement learning tasks, with over-approximation shown in *grey* background (marked in subcolumns **ox**) and under-approximation in *white* background (marked in subcolumns **ux**).

Task	**Property**	**Config**	**#Poly**		**Cov**		**Time(s)**	
			ux	**ox**	**ux**	**ox**	**ux**	**ox**
Cartpole (FNN 2×64)	$\{y \in \mathbb{R}^2 \vert\ y_1 \geq y_2\}$	$\dot{\theta} \in [-2, -1]$	25	1	0.766	1.213	13.337	2.149
		$\dot{\theta} \in [-2, -0.5]$	42	8	0.750	1.242	19.732	5.778
		$\dot{\theta} \in [-2, 0]$	66	22	0.755	1.246	30.563	11.476
Lunarlander (FNN 2×64)	$\{y \in \mathbb{R}^4 \vert \wedge_{i\in\{1,3,4\}} y_2 \geq y_i\}$	$\dot{v} \in [-1, 0]$	18	1	0.754	1.068	14.453	2.381
		$\dot{v} \in [-2, 0]$	67	23	0.751	1.246	48.455	19.210
		$\dot{v} \in [-4, 0]$	97	90	0.751	1.249	76.234	72.285
Dubinsrejoin (FNN 2×256)	$\{y \in \mathbb{R}^8 \vert \wedge_{i\in[2,4]}\ y_1 \geq y_i \bigwedge \wedge_{i\in[6,8]}\ y_5 \geq y_i\}$	$x_v \in [-0.1, 0.1]$	211	20	0.751	1.242	182.821	18.666
		$x_v \in [-0.2, 0.2]$	409	23	0.750	1.241	323.839	24.788
		$x_v \in [-0.3, 0.3]$	677	43	0.750	1.244	589.939	41.502

computation time for property $O = \{y \in \mathbb{R}^9 \ \vert \wedge_{i\in[1,9]\setminus 3}\ y_3 \geq y_i\}$, with the maximum reaching 191 polytopes and computation time of 107.758s for VCAS model 3.

Neural Network Controllers. In this experiment, we consider preimage approximation for neural network controllers in reinforcement learning tasks. Note that the *Exact* method in Matoba and Fleuret (2020) is unable to deal with neural networks of these sizes and Invprop in Kotha et al. (2023) is not capable of characterising the preimage under-approximation in the form of disjoint polytopes. Table 3 summarises the experimental results obtained by our method, where the columns for over-approximations are marked with grey background and under-approximations marked with a white background.

We evaluate Algorithm 2 (with input splitting) with respect to a range of different configurations of the input region (e.g., angular velocity $\dot{\theta}$ for Cartpole). For comparison, we set the same target coverage ratio for different input region sizes (0.75 for under-approximation and 1.25 for over-approximation) and an iteration limit of 1000. In Table 3, we see that our method successfully generates preimage approximations for all configurations, reaching the targeted approximation coverage. Empirically, for the same coverage ratio, our method

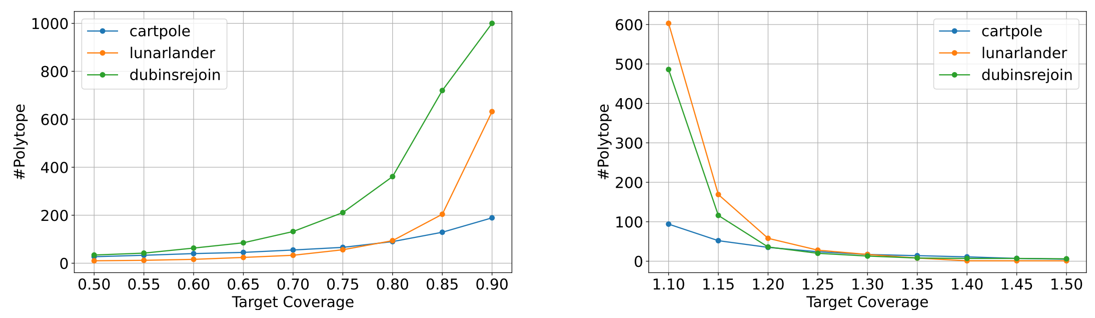


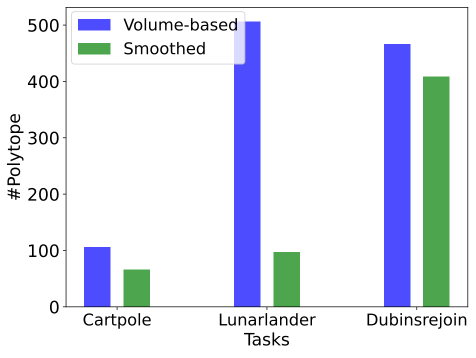
Figure 10: Number of polytopes in preimage approximation for a range of target coverage values. Left: under-approximation. Right: over-approximation.

requires a number of polytopes and time roughly linear in the input region size for the preimage under-approximation. For over-approximations, the bounding constraints of the input region are added as additional constraints to form the polytope approximation on each subregion, which affects the linear trend in the number of polytopes and computation time as the input region size increases. For example, the constraint brought by the input configuration of $-2 \leq \dot{\theta} \leq -1$ for Cartpole, together with the single polytope over-approximation, already reaches the target coverage, while for a larger input region of $-2 \leq \dot{\theta} \leq 0$, 22 preimage polytopes are needed together with input bounding constraints for each input subregion. Table 3 also shows that the number of polytopes, and consequently the runtime cost, required for computing over-approximations is lower than for under-approximations. This difference can be attributed to how half-plane constraints are used to form the polytope. For over-approximation, the input bounds of each subregion act as additional half-plane constraints that further tighten the feasible set, resulting in fewer polytopes that reach the over-approximation precision. In contrast, the under-approximation imposes more strict constraints than the subregion bounds, as it is guaranteed to be included in the subregion. This often requires a finer splitting of the input space to achieve the target precision, resulting in a greater number of polytopes and increased computational effort.

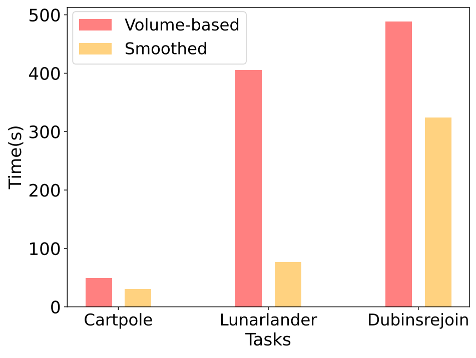
In Figure 10, we show the number of polytopes needed to reach different target coverage ratios for both under-approximation (left) and over-approximation (right). Our evaluation results indicate that the number of refinement iterations taken is influenced by the number of output constraints and the size of the neural network. For instance, the neural network controller for Cartpole, which has a single output constraint, shows a roughly linear increase in the number of polytopes as the target coverage increases for under-approximation (resp. decreases for over-approximation). In contrast, accommodating multiple output constraints for larger neural networks, e.g., Dubinsrejoin, requires a significant increase in refinement iterations as the target coverage approaches 1.

6.2.2 Effectiveness of Smoothed Input Splitting

We now analyse the effectiveness of the smoothed splitting method described in Section 5.3 (Equation 8 and 9), in comparison to a volume-guided splitting method in Zhang et al. (2024b) that chooses the input feature leading to the greatest improvement in approximation

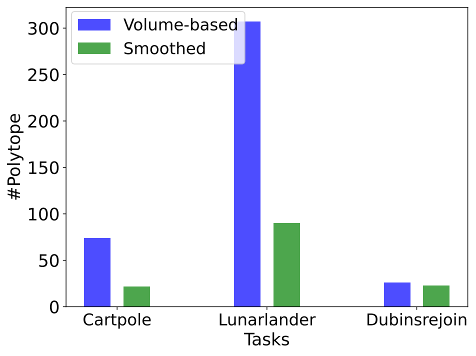


(a) Number of polytopes

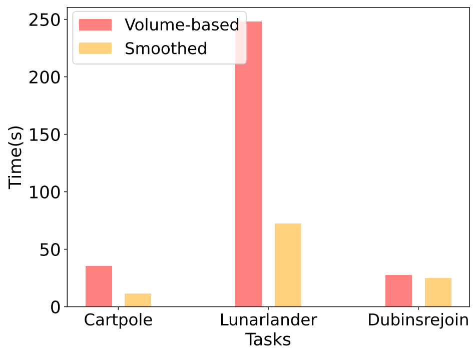


(b) Time cost

Figure 11: Effectiveness of smooth splitting for preimage under-approximation. Comparison results with Volume-based method from Zhang et al. (2024b).



(a) Number of polytopes



(b) Time cost

Figure 12: Effectiveness of smooth splitting for preimage over-approximation. Comparison results with Volume-based method from Zhang et al. (2024b).

volume. From Figures 11 and 12, we observe that the smoothed splitting method requires significantly fewer refinement iterations for all reinforcement learning controllers to achieve the target coverage, thus reducing the number of polytopes and computation time, than the volume-guided splitting method. More specifically, the smoothed splitting method achieves an average reduction of 43.6% in the number of polytopes and 51.0% in computation time for under-approximation across the neural network controllers, up to 80.8%/81.2% reduction for the Lunarlander task. Similar improvements in computation efficiency and size of polytope union are also achieved for over-approximations, with an average reduction of 50.8%/49.6% across all reinforcement learning tasks.

Recall that the smoothed input splitting heuristic relaxes the volume-based heuristic, such that, for each sampled input point in the input region, we take into account not only *whether* the point lies in the polytope approximation, but also *how far away* the point is from the approximation. This is particularly crucial in early iterations, where the approximation

Table 4: Preimage under-approximation refinement with ReLU split (L_∞ attack). Results with Lagrangian optimisation are marked in grey background in columns *w/ LagOpt*.

L_∞ attack (FNN 6×100)	**#Poly**		**Cov**		**Time(s)**	
	w/o	**w/ LagOpt**	**w/o**	**w/ LagOpt**	**w/o**	**w/ LagOpt**
0.06	2	2	1.0	1.0	3.183	3.237
0.07	247	40	0.752	0.756	130.746	29.019
0.08	522	290	0.751	0.751	305.867	218.455
0.09	733	563	0.165	0.751	507.116	365.552

Table 5: Preimage under-approximation refinement with ReLU split (patch attack). Results with Lagrangian optimisation are marked in grey background in columns *w/ LagOpt*.

Patch attack (FNN 6×100)	**#Poly**		**Cov**		**Time(s)**	
	w/o	**w/ LagOpt**	**w/o**	**w/ LagOpt**	**w/o**	**w/ LagOpt**
3×3(center)	1	1	1.0	1.0	2.611	2.637
4×4(center)	678	678	0.382	0.427	455.988	514.272
7×7(corner)	7	7	0.842	0.861	6.065	6.217
8×7(corner)	956	954	0.033	0.214	488.849	676.666

may be too loose; for example, an under-approximation may have no overlap with the input region (thus zero volume). Therefore, computing the approximation volume (after splitting on each input feature) provides very little signal. In such cases, the smoothed splitting heuristic is able to capture promising input features that, while not immediately improving the approximation volume, can bring the preimage bounding planes closer to the exact preimage, which is beneficial for future iterations.

6.2.3 Effectiveness of Preimage Approximation with ReLU Split

In this subsection, we evaluate the scalability of Algorithm 2 with ReLU splitting by applying it to MNIST image classifiers. In particular, we consider input regions defined by bounded perturbations to a given MNIST image. Table 4 and 5 summarise the evaluation results for two types of image perturbations commonly considered in the adversarial robustness literature (L_∞ and patch attack, respectively). For L_∞ attacks, bounded perturbation noise is applied to all image pixels. The patch attack applies only to a smaller patch area of $n \times m$ pixels but allows arbitrary perturbations covering the whole valid range $[0, 1]$. The task is then to produce a DUP approximation of the subset of the perturbation region that is guaranteed to be classified correctly.

For L_∞ attack, we evaluate our method over perturbations of increasing size, from 0.06 to 0.09. It is worth noting that for this size of preimage, e.g., from 0.06 to 0.07, the *volume* of the input region increases by tens of orders of magnitude due to the high dimensionality, making effective preimage approximation significantly more challenging. Table 4 shows that our approach (Algorithm 2) without Lagrangian optimisation (marked in columns *w/o*) is able to generate a preimage under-approximation that achieves the targeted coverage of 0.75 for L_∞ noise up to 0.08. The fact that the number of polytopes and computation time remain manageable is due to the effectiveness of ReLU splitting. In Table 5, for the patch attack, we observe that the number of polytopes and time required increase sharply when

Table 6: Preimage over-approximation refinement with ReLU split (patch attack). Results with Lagrangian optimisation are marked in grey background in columns *w/ LagOpt*.

Patch attack (FNN 6×100)	**#Poly**		**Cov**		**Time(s)**	
	w/o	**w/ LagOpt**	**w/o**	**w/ LagOpt**	**w/o**	**w/ LagOpt**
10×10(center)	387	387	1.099	1.099	261.826	281.916
11×11(center)	317	317	1.249	1.249	192.954	212.735
16×15(corner)	616	616	1.050	1.050	328.589	350.092
16×16(corner)	285	285	1.249	1.249	165.250	175.605

increasing the patch size for both the centre and corner area of the image, suggesting that the model is more sensitive to larger local perturbations. It is also interesting that our method can generate preimage approximations for larger patches in the corner as opposed to the centre of the image; we hypothesize this is due to the greater influence of central pixels on the neural network output, and correspondingly a greater number of unstable neurons over the input perturbation space.

Table 6 shows the preimage refinement results for over-approximations in the context of patch attack. The results of our approach (Algorithm 2) without Lagrangian optimisation are summarised in columns *w/o*. As shown in the table, our refinement method can effectively tighten the over-approximation to the targeted coverage of 1.25 for different attack configurations. For patch size 10×10 (centre) and 16×15 (corner), we found that the perturbation region is a trivial over-approximation itself for the target coverage of 1.25; thus, we demonstrate the results with a target coverage of 1.1 and 1.05. Similarly to under-approximations, a patch attack in the centre with a smaller patch size requires more refinement iterations than the patch attack in the corner, demonstrating a greater influence of central pixels.

Effectiveness of Lagrangian Optimisation. The results of evaluation of our approach (Algorithm 2) for under-approximation with Lagrangian optimisation are shown in Table 4 and 5 (marked in columns *w/ LagOpt* with grey background). For L_∞ attack, the refinement method with Lagrangian optimisation generates preimage approximations that achieve the target coverage of 0.75 for all perturbation settings, including perturbation noise 0.09 where the refinement *without* Lagrangian optimisation fails (0.751 vs 0.165 in Table 4). The new refinement method also leads to a significant reduction in the number of polytopes and computation cost. For the patch attack, the refinement method with Lagrangian optimisation effectively improves the preimage approximation precision for all configuration settings. Since the patch attack allows arbitrary perturbations covering the whole valid range $[0, 1]$, it leads to a rapid increase in the number of unstable neurons and exhausts the iteration limit when increasing the patch size. Nonetheless, the resulting preimage approximation coverage obtained with Lagrangian optimisation shows better per-iteration precision improvement, while introducing marginal computation overhead compared to the previous method.

Columns *w/ LagOpt* in Table 6 summarises the over-approximation results with Lagrangian optimisation. In this case, we introduce the Lagrange multipliers with the opposite signs to the under-approximation to guarantee the validity of the symbolic over-approximation. Intriguingly, in contrast to under-approximation, we find that the optimised β parameters are almost always close to 0, meaning that the results are similar to

Table 7: Comparison with a robustness verifier.

Task	**α, β-CROWN**		**PREMAP**		
	Result	Time	Cov(%)	#Poly	Time
Cartpole ($\dot{\theta} \in [-1.642, -1.546]$)	yes	3.349	100.0	1	1.137
Cartpole ($\dot{\theta} \in [-1.642, 0]$)	no	6.927	94.9	2	3.632
MNIST (L_∞ 0.026)	yes	3.415	100.0	1	2.649
MNIST (L_∞ 0.04)	unknown	267.139	100.0	2	3.019

not using Lagrangian optimisation. We hypothesize that, for over-approximations, the objective function is relatively flat in the vicinity of 0, which makes the parameters difficult to optimise.

Comparison with Robustness Verifiers. We now illustrate empirically the utility of preimage computation in robustness analysis compared to robustness verifiers. Table 7 shows comparison results with α, β-CROWN, winner of the VNN competition (Müller et al., 2022). We set the tasks according to the problem instances from VNN-COMP 2022 for local robustness verification (localised perturbation regions). For Cartpole, α, β-CROWN can provide a verification guarantee (yes/no or safe/unsafe) for both problem instances. However, in the case where the robustness property does not hold, our method explicitly generates a preimage under-approximation in the form of a disjoint polytope union (which guarantees the satisfaction of the output properties), and covers 94.9% of the exact preimage. For MNIST, while the smaller perturbation region is successfully verified, α, β-CROWN with tightened intermediate bounds by MIP solvers returns unknown with a timeout of 300s for the larger region. In comparison, our algorithm provides a concrete union of polytopes where the input is guaranteed to be correctly classified, which we find covers 100% of the input region (up to sampling error). Note also, as shown in Table 4, our algorithm can produce non-trivial under-approximations for input regions far larger than α, β-CROWN can verify.

Table 8: Performance comparison on preimage generation for reinforcement learning tasks. Under-approximation results are in the column labelled by Under-Approx, whereas over-approximation is in the column labelled by Over-Approx.

Task	**Method**	**Under-Approx**			**Over-Approx**		
		#Poly	**Cov**	**Time(s)**	**#Poly**	**Cov**	**Time(s)**
Cartpole	Invprop	81	0.760	19.835	196	**1.119**	224.718
	PREMAP	**25**	**0.766**	**13.337**	**8**	1.242	**5.778**
Lunarlander	Invprop	256	0.756	325.206	221	1.378	358.386
	PREMAP	**12**	**0.759**	**8.091**	**90**	**1.247**	**50.154**
Dubinsrejoin	Invprop	295	0.441	360.69	205	1.876	361.16
	PREMAP	**20**	**0.765**	**12.683**	**13**	**1.232**	**8.822**

6.2.4 Comparison with Output Constraint-Enhanced Approach

The original Invprop method focused on preimage over-approximation in low-dimensional tasks. In our experiments, we compare our method, PREMAP, with the updated version of Invprop, which is now integrated into the auto-LiRPA framework, enabling both under- and over-approximation, which we still refer to as Invprop. Table 8 presents a comparative evaluation of preimage approximation quality on neural network controllers between Invprop and our proposed method, PREMAP. Note that, for Invprop, output constraints are leveraged to compute tighter intermediate bounds, thereby improving the preimage approximation quality.

The evaluation results show that PREMAP demonstrates superior preimage approximation precision and runtime efficiency across all evaluated neural network controllers. The results highlight the importance of effective preimage refinement strategies. Specifically, PREMAP's effectiveness relies on the smart subregion selection and greedy splitting, which leads to better preimage quality improvement than solely applying intermediate bound tightening on subregions. A promising direction for future work is to explore a hybrid approach that integrates PREMAP's domain selection and splitting method with the bound-tightening capabilities of Invprop. Investigating whether this synergy can consistently outperform each method individually could offer valuable insights to advance the effectiveness of preimage analysis techniques.

6.2.5 Evaluation of Input vs ReLU Split

Table 9 presents performance comparison between input and ReLU splitting across low-dimensional control tasks. Both approaches are able to reach the target coverage; however, input splitting generally achieves higher per-iteration precision refinement. It requires fewer iterations, and as a result, fewer preimage polytopes compared to ReLU splitting. This advantage is mainly because one input split of the original region into smaller ones can simultaneously stabilise multiple (unstable) ReLU neurons, whereas ReLU splitting addresses one unstable neuron per iteration.

Notably, the proposed greedy input splitting method further selects the input feature that yields the greatest improvement in approximation precision per split, though at the cost of increased computational overhead in certain cases (e.g., Lunarlander). For the same reason, for high-dimensional tasks such as the MNIST classification task with 784 input features, the same greedy method requires parallel computation across hundreds of processes. Each instance involves large intermediate tensors, which can easily exceed the GPU memory capacity, and the greedy selection for the optimal split further increases computation time.

6.2.6 Quantitative Verification

We now demonstrate the application of our preimage under-approximation to quantitative verification of the property (I, O, p); that is, we aim to check whether $f(x) \in O$ for at least proportion p of input values $x \in I$. Table 10 summarises the quantitative verification results, which leverage the disjointness of our under-approximation, such that we can compute the total volume covered by computing the volume of each individual polytope.

Table 9: Preimage under-approximation refinement with Input vs ReLU split.

Task	**Config**	**#Poly**		**Cov(%)**		**Time(s)**	
		Input	ReLU	Input	ReLU	Input	ReLU
Cartpole	$\dot{\theta} \in [0, 0.1]$	**3**	106	**78.0**	75.3	**4.233**	12.613
Lunarlander	$\dot{v} \in [-0.1, 0]$	**36**	134	**75.1**	**75.1**	35.055	**14.239**
Dubinsrejoin	$x_v \in [-0.1, 0]$	**2**	226	**94.5**	75.0	**3.638**	34.895

Table 10: Quantitative verification results with preimage under-approximation.

Task	**Property**	**#Poly**	**Time(s)**	**QuantProp(%)**
VCAS	$O = \{y \in \mathbb{R}^9 \vert \bigwedge_{i=2}^{9} y_1 - y_i \geq 0\}$	6	5.620	90.8
Cartpole	$O = \{y \vert y_1 - y_2 \geq 0\}$	11	12.1	90.0
Lunarlander	$O = \{y \in \mathbb{R}^4 \vert \wedge_{i \in \{1,3,4\}} y_2 \geq y_i\}$	120	429.480	90.0

Vertical Collision Avoidance System. In this example, we consider the VCAS system and a scenario where the two aircraft have negative relative altitude from intruder to ownship ($h \in [-8000, 0]$), the ownship aircraft has a positive climbing rate $\dot{h}_A \in [0, 100]$ and the intruder has a stable negative climbing rate $\dot{h}_B = -30$, and time to the loss of horizontal separation is $t \in [0, 40]$, which defines the input region I. For this scenario, the correct advisory is "Clear Of Conflict" (COC). We apply Algorithm 3 to verify the quantitative property where $O = \{y \in \mathbb{R}^9 | \bigwedge_{i=2}^{9} y_1 - y_i \geq 0\}$ and the proportion $p = 0.9$, with an iteration limit of 1000. The quantitative proportion reached by the generated under-approximation is 90.8%, which verifies the quantitative property in 5.620s.

Cartpole. In the Cartpole problem, the objective is to balance the pole attached to a cart by pushing the cart either left or right. We consider a scenario where the cart position is to the right of the centre ($x \in [0, 1]$), the cart is moving right ($\dot{x} \in [0, 0.5]$), the pole is slightly tilted to the right ($\theta \in [0, 0.1]$) and pole is moving to the left ($\dot{\theta} \in [-0.2, 0]$). To balance the pole, the neural network controller needs to determine "pushing left". We apply Algorithm 3 to verify the quantitative property, where $O = \{y | y_1 - y_2 \geq 0\}$ and the proportion $p = 0.9$, with an iteration limit of 1000. The under-approximation algorithm takes 12.1s to reach the target proportion 90.0%.

Lunarlander. In the Lunarlander task, the objective of the neural networks controller is to achieve a safe landing of the lander. Consider a scenario where the lander is slightly to the left of the centre of the landing pad ($x \in [-1, 0]$), the lander is above the landing pad sufficient for descent correction ($h \in [0, 1]$), and it is moving to the right ($\dot{x} \in [1, 2]$) but descending rapidly ($\dot{h} \in [-2, -1]$). To avoid a hard landing, the neural network controller needs to reduce the descent speed by taking the action "fire main engine". We formulate the quantitative property for this task, where $O = \{y \in \mathbb{R}^4 | \wedge_{i \in \{1,3,4\}} y_2 \geq y_i\}$ and the proportion $p = 0.9$. To compute preimage under-approximation for this more complex task takes 429.480s to reach the target proportion 90.0%.

7. Conclusion

We present PREMAP, an efficient and unifying algorithm for preimage approximation of neural networks. Our *anytime* method stems from the observation that linear relaxation can be used to efficiently produce approximations, in conjunction with custom-designed strategies for iteratively decomposing the problem to rapidly improve the approximation quality. We formulate the preimage approximation in each refinement iteration as an optimisation problem and propose a differentiable objective to derive tighter preimages via optimising over convex bounding parameters and Lagrange multipliers. Unlike previous approaches, our method is designed for, and scales to, both low and high-dimensional problems. Experimental evaluation on a range of benchmark tasks shows significant advantages in runtime efficiency and scalability, and the utility of our method for important applications in quantitative verification and robustness analysis.

Acknowledgments

This project received funding from the ERC under the European Union's Horizon 2020 research and innovation programme (FUN2MODEL, grant agreement No. 834115) and ELSA: European Lighthouse on Secure and Safe AI project (grant agreement No. 101070617 under UK guarantee). This work was done in part while Benjie Wang was visiting the Simons Institute for the Theory of Computing.

Appendix A. Experiment Setup

In this section, we present the detailed configuration of neural networks in the benchmark tasks.

A.1 Vehicle Parking.

For the vehicle parking task, we train a neural network with one hidden layer of 20 neurons, which is computationally feasible for exact preimage computation for comparison. We consider computing the preimage approximation with input region corresponding to the entire input space $\mathcal{C} = \{x \in \mathbb{R}^2 | x \in [0, 2]^2\}$, and output sets O_k, which correspond to the neural network outputting label k: $O_k = \{y \in \mathbb{R}^4 \mid \bigwedge_{i \in \{1,2,3,4\} \backslash k} y_k - y_i \geq 0\}, \quad k \in \{1, 2, 3, 4\}$.

A.2 Aircraft Collision Avoidance

The aircraft collision avoidance (VCAS) system (Julian and Kochenderfer, 2019) is used to provide advisory for collision avoidance between the ownship aircraft and the intruder. VCAS uses four input features $(h, \dot{h}_A, \dot{h}_B, t)$ representing the relative altitude of the aircrafts, vertical climbing rates of the ownship and intruder aircrafts, respectively, and time to the loss of horizontal separation. VCAS is implemented by nine feed-forward neural networks built with a hidden layer of 21 neurons. In our experiment, we use the following input region for the ownship and intruder aircraft as in Matoba and Fleuret (2020): $h \in [-8000, 8000]$, $\dot{h}_A \in [-100, 100]$, $\dot{h}_B = 30$, and $t \in [0, 40]$. In the training, the input configurations are normalized into a range of $[-1, 1]$. We consider the output property $O = \{y \in \mathbb{R}^9 \mid \wedge_{i\in[2,9]} \; y_1 \geq y_i\}$ and generate the preimage approximation for the VCAS neural networks.

A.3 Neural Network Controllers

A.3.1 Cartpole

The cartpole control problem considers balancing a pole atop a cart by controlling the movement of the cart. The neural network controller has two hidden layers with 64 neurons, and uses four input variables representing the position and velocity of the cart, the angle and angular velocity of the pole. The controller outputs are pushing the cart left or right. In the experiments, we set the following input region for the Cartpole task: (1) cart position $[-1, 1]$, (2) cart velocity $[0, 2]$, (3) angle of the pole $[-0.2, 0]$, and (4) angular velocity of the pole $[-2, 0]$ (with varied feature length in the evaluation). We consider the output property for the action *pushing left*.

A.3.2 Lunarlander

The Lunarlander problem considers the task of correct landing of a moon lander on a landing pad. The neural network for Lunarlander has two hidden layers with 64 neurons, and eight input features addressing the lander's coordinate, orientation, velocities, and ground contact indicators. The outputs represent four actions. For the Lunarlander task, we set the input region as: (1) horizontal and vertical position $[-1, 0] \times [0, 1]$, (2) horizontal and vertical velocity $[0, 2] \times [-2, 0]$ (with varied feature length for evaluation), (3) angle and angular velocity $[-1, 0] \times [-0.1, 0.1]$, (4) left and right leg contact $[0.9, 1]^2$. We consider the output specification for the action "fire main engine", i.e., $\{y \in \mathbb{R}^4 \mid \wedge_{i\in\{1,3,4\}} y_2 \geq y_i\}$.

A.3.3 Dubinsrejoin.

The Dubinsrejoin problem considers guiding a wingman craft to a certain radius around a lead aircraft. The neural network controller has two hidden layers with 256 neurons. The input space of the neural network controller is eight-dimensional, with the input variables capturing the position, heading, velocity of the lead and wingman crafts, respectively. The outputs are also eight dimensional representing controlling actions of the wingman. Note that the eight neural network outputs are processed further as tuples of actuators (rudder, throttle) for controlling the wingman where each actuator has 4 options. The control action tuple is decided by taking the action with the maximum output value among the first four network outputs (the first actuator options) and the action with the maximum value among the second four network outputs (the second actuator options). In the experiments, we set the following input region: (1) horizontal and vertical position $[-0.2, 0] \times [0, 0.5]$, (2) heading and velocity $[-1, 0] \times [0, 0.2]$ for the lead aircraft, and (3) horizontal and vertical position $[0.4, 0.6] \times [-0.3, 0.3]$ (with varied feature length for evaluation), (4) heading and velocity $[0.2, 0.5] \times [-0.5, 0.5]$ for the wingman aircraft. We consider the output property that both actuators (rudder, throttle) take the first option, i.e., $\{y \in \mathbb{R}^8 \mid \wedge_{i\in\{2,3,4\}} \; y_1 \geq y_i \bigwedge \wedge_{i\in\{6,7,8\}} \; y_5 \geq y_i\}$.

A.4 MNIST Classification

We use the trained neural network from VNN-COMP 2022 (Müller et al., 2022) for digit image classification. The neural network has six layers with a hidden neuron size of 100 for each hidden layer. We consider two types of image attacks: l_∞ and patch attack. For L_∞ attack, a perturbation is applied to all pixels of the image. For the patch attack, it applies arbitrary perturbations to the patch area, i.e., the perturbation noise covers the whole valid range $[0, 1]$, for which we set the patch area at the centre and (upper-left) corner of the image with different sizes.

Appendix B. Proofs

We present the propositions and proofs on guaranteed polytope volume improvement with each refinement iteration, noting that these propositions are valid without stochastic optimisation. Subsequently, we provide proofs for Propositions 7 and 8.

Proposition 9 *Given any subregion $\mathcal{C}_{sub}$ with polytope under-approximation $T_{\mathcal{C}_{sub}}(O)$, and its children $\mathcal{C}^l_{sub}, \mathcal{C}^u_{sub}$ with polytope under-approximations $T_{\mathcal{C}^l_{sub}}(O), T_{\mathcal{C}^u_{sub}}(O)$ respectively, it holds that:*

$$T_{\mathcal{C}^l_{sub}}(O) \cup T_{\mathcal{C}^u_{sub}}(O) \supseteq T_{\mathcal{C}_{sub}}(O) \tag{24}$$

Proof We define $T_{\mathcal{C}_{sub}}(O)|_l, T_{\mathcal{C}_{sub}}(O)|_r$ to be the restrictions of $T_{\mathcal{C}_{sub}}(O)$ to $\mathcal{C}^l_{sub}$ and $\mathcal{C}^r_{sub}$ respectively, that is:

$$T_{\mathcal{C}_{sub}}(O)|_l = \{x \in \mathbb{R}^d | \bigwedge_{i=1}^{K} (\underline{g_i}(x) \geq 0) \wedge (x \in \mathcal{C}^l_{sub})\} \tag{25}$$

$$T_{\mathcal{C}_{sub}}(O)|_r = \{x \in \mathbb{R}^d | \bigwedge_{i=1}^{K} (\underline{g_i}(x) \geq 0) \wedge (x \in \mathcal{C}^r_{sub})\} \tag{26}$$

where we have replaced the constraint $x \in \mathcal{C}_{sub}$ with $x \in \mathcal{C}^l_{sub}$ (resp. $x \in \mathcal{C}^r_{sub}$), and $\underline{g_i}(x)$ is the LiRPA lower bound for the i^{th} specification on the input region $\mathcal{C}_{sub}$.

On the other hand, we also have:

$$T_{\mathcal{C}^l_{sub}}(O) = \{x \in \mathbb{R}^d | \bigwedge_{i=1}^{K} (\underline{g_{l,i}}(x) \geq 0) \wedge (x \in \mathcal{C}^l_{sub})\} \tag{27}$$

$$T_{\mathcal{C}^r_{sub}}(O) = \{x \in \mathbb{R}^d | \bigwedge_{i=1}^{K} (\underline{g_{r,i}}(x) \geq 0) \wedge (x \in \mathcal{C}^r_{sub})\} \tag{28}$$

where $\underline{g_{l,i}}(x)$ (resp. $\underline{g_{r,i}}(x)$) is the LiRPA lower bound for the i^{th} specification on the input region $\mathcal{C}^l_{sub}$ (resp. $\mathcal{C}^r_{sub}$). Now, it is sufficient to show that $T_{\mathcal{C}^l_{sub}}(O) \supseteq T_{\mathcal{C}_{sub}}(O)|_l$ and $T_{\mathcal{C}^r_{sub}}(O) \supseteq T_{\mathcal{C}_{sub}}(O)|_r$ to prove Equation 24. We will now show that $T_{\mathcal{C}^l_{sub}}(O) \supseteq T_{\mathcal{C}_{sub}}(O)|_l$ (the proof for $T_{\mathcal{C}^r_{sub}}(O) \supseteq T_{\mathcal{C}_{sub}}(O)|_r$ is entirely similar).

Before proving this result in full, we outline the approach and a sketch proof. It suffices to prove (for all i) that $\underline{g_{l,i}}(x)$ is a tighter bound than $\underline{g_i}(x)$ on $\mathcal{C}^l_{sub}$. That is, to show that

$\underline{g_{l,i}}(x) \geq \underline{g_i}(x)$ for inputs x in $\mathcal{C}^l_{sub}$, as then $\underline{g_i}(x) \geq 0 \implies \underline{g_{l,i}}(x) \geq 0$ for inputs x in $\mathcal{C}^l_{sub}$, and so $T_{\mathcal{C}^l_{sub}}(O) \supseteq T_{\mathcal{C}_{sub}}(O)|_l$. The bound $\underline{g_{l,i}}(x)$ is tighter than $\underline{g_i}(x)$ because the input region for LiRPA is smaller for $\underline{g_{l,i}}(x)$, leading to tighter concrete neuron bounds, and thus tighter bound propagation through each layer of the neural network g_i. We present the formal proof of greater bound tightness for input and ReLU splitting in the following.

Input split: We show $\underline{g_{l,i}}(x) \geq \underline{g_i}(x)$ for all $x \in \mathcal{C}^l_{sub}$ by induction (dropping the index i in the following as it is not important). Recall that LiRPA generates symbolic upper and lower bounds on the pre-activation values of each layer in terms of the input (i.e. treating that layer as output), which can then be converted into concrete bounds.

$$\underline{\mathbf{A}}^{(j)}x + \underline{\mathbf{b}}^{(j)} \leq h^{(j)}(x) \leq \overline{\mathbf{A}}^{(j)}x + \overline{\mathbf{b}}^{(j)} \tag{29}$$

$$\underline{\mathbf{A}}^{(l,j)}x + \underline{\mathbf{b}}^{(l,j)} \leq h^{(j)}(x) \leq \overline{\mathbf{A}}^{(l,j)}x + \overline{\mathbf{b}}^{(l,j)} \tag{30}$$

where $h^{(j)}(x)$ are the pre-activation values for the j^{th} layer of the network g_i, and $\underline{\mathbf{A}}^{(j)}, \underline{\mathbf{b}}^{(j)}, \overline{\mathbf{A}}^{(j)}, \overline{\mathbf{b}}^{(j)}$ (resp. $\underline{\mathbf{A}}^{(l,j)}, \underline{\mathbf{b}}^{(l,j)}, \overline{\mathbf{A}}^{(l,j)}, \overline{\mathbf{b}}^{(l,j)}$) are the linear bound coefficients, for input regions $\mathcal{C}_{sub}$ (resp. $\mathcal{C}^l_{sub}$).

Inductive Hypothesis For all layers $j = 1, ..., L$ in the network, and for all $x \in \mathcal{C}^l_{sub}$, it holds that:

$$\underline{\mathbf{A}}^{(j)}x + \underline{\mathbf{b}}^{(j)} \leq \underline{\mathbf{A}}^{(l,j)}x + \underline{\mathbf{b}}^{(l,j)} \leq \overline{\mathbf{A}}^{(l,j)}x + \overline{\mathbf{b}}^{(l,j)} \leq \overline{\mathbf{A}}^{(j)}x + \overline{\mathbf{b}}^{(j)} \tag{31}$$

Base Case For the input layer, we have the trivial bounds $\mathbf{I}x \leq x \leq \mathbf{I}x$ for both regions.

Inductive Step Suppose that the inductive hypothesis is true for layer $j-1 < L$. Using the symbolic bounds in Equations 29, 30, we can derive concrete bounds $\mathbf{l}^{(j-1)} \leq h^{(j-1)}(x) \leq \mathbf{u}^{(j-1)}$ and $\mathbf{l}^{(l,j-1)} \leq h^{(l,j-1)}(x) \leq \mathbf{u}^{(l,j-1)}$ on the values of the pre-activation layer. By the inductive hypothesis, the bounds for region $\mathcal{C}^l_{sub}$ will be tighter, i.e. $\mathbf{l}^{(j-1)} \leq \mathbf{l}^{(l,j-1)} \leq \mathbf{u}^{(l,j-1)} \leq \mathbf{u}^{(j-1)}$. Now, consider the backward bounding procedure for layer j as output. We begin by encoding the linear layer from post-activation layer $j-1$ to pre-activation layer j as:

$$\mathbf{W}^{(j)}a^{(j-1)}(x) + \mathbf{b}^{(j)} \leq h^{(j)}(x) \leq \mathbf{W}^{(j)}a^{(j-1)}(x) + \mathbf{b}^{(j)} \tag{32}$$

Then, we bound $a^{(j-1)}(x)$ in terms of $h^{(j-1)}(x)$ using linear relaxation. Consider the three cases in Figure 1 (reproduced from main paper), where we have a bound $\underline{c}h^{(j-1)}_k(x) + \underline{d} \leq a^{(j-1)}_k(x) \leq \overline{c}h^{(j-1)}_k(x) + \overline{d}$, for some scalars $\underline{c}, \underline{d}, \overline{c}, \overline{d}$. If the concrete bounds (horizontal axis) are tightened, then an unstable neuron may become inactive or active, but not vice versa. It can thus be seen that the new linear upper and lower bounds on $h^{(j-1)}_k(x)$ will also be tighter.

Substituting the linear relaxation bounds in Equation 32 as in Xu et al. (2021), we obtain bounds of the form

$$\underline{\mathbf{A}}^{(j)}_j h^{(j-1)}(x) + \underline{\mathbf{b}}^{(j)}_j \leq h^{(j)}(x) \leq \overline{\mathbf{A}}^{(j)}_j h^{(j-1)}(x) + \overline{\mathbf{b}}^{(j)}_j \tag{33}$$

$$\underline{\mathbf{A}}^{(l,j)}_j h^{(j-1)}(x) + \underline{\mathbf{b}}^{(l,j)}_j \leq h^{(j)}(x) \leq \overline{\mathbf{A}}^{(l,j)}_j h^{(j-1)}(x) + \overline{\mathbf{b}}^{(l,j)}_j \tag{34}$$

such that $\underline{\mathbf{A}}_j^{(j)} h^{(j-1)}(x) + \underline{\mathbf{b}}_j^{(j)} \le \underline{\mathbf{A}}_j^{(l,j)} h^{(j-1)}(x) + \underline{\mathbf{b}}_j^{(l,j)} \le \overline{\mathbf{A}}_j^{(l,j)} h^{(j-1)}(x) + \overline{\mathbf{b}}_j^{(l,j)} \le \overline{\mathbf{A}}_j^{(j)} h^{(j-1)}(x) + \overline{\mathbf{b}}_j^{(j)}$ for all $\mathbf{l}^{(l,j-1)} \le h^{(j-1)}(x) \le \mathbf{l}^{(l,j-1)}$, by the fact that the concrete bounds are tighter for $\mathcal{C}_{sub}^l$.

Finally, substituting the bounds in Equations 29 and 30 (for $h^{(j-1)}$), and using the tightness result in the inductive hypothesis for $j-1$, we obtain linear bounds for $h^{(j)}(x)$ in terms of of the input x, such that the inductive hypothesis for j holds.

ReLU split: We use $\mathcal{C}_{sub}^l$ and $\mathcal{C}_{sub}^r$ to denote the input subregions when fixing unstable ReLU neuron $z_k^{(j-1)} = h_k^{(j-1)}(x)$, i.e., $\mathcal{C}_{sub}^l = \{x \mid h_k^{(j-1)}(x) \ge 0\}$ and $\mathcal{C}_{sub}^r = \{x \mid h_k^{(j-1)}(x) < 0\}$.

In the following, we prove that $\underline{g_{l,i}}(x) \ge \underline{g_i}(x)$ for all $x \in \mathcal{C}_{sub}^l$. Assume we fix one unstable ReLU neuron of layer $j-1$, then for all layers $1 \le m \le j-1$, for all $x \in \mathcal{C}_{sub}^l$, it holds that:

$$\underline{\mathbf{A}}^{(m)} x + \underline{\mathbf{b}}^{(m)} \le \underline{\mathbf{A}}^{(l,m)} x + \underline{\mathbf{b}}^{(l,m)} \le \overline{\mathbf{A}}^{(l,m)} x + \overline{\mathbf{b}}^{(l,m)} \le \overline{\mathbf{A}}^{(m)} x + \overline{\mathbf{b}}^{(m)} \tag{35}$$

where $\underline{\mathbf{A}}^{(m)} = \underline{\mathbf{A}}^{(l,m)}$, $\underline{\mathbf{b}}^{(m)} = \underline{\mathbf{b}}^{(l,m)}$ and same for the upper bounding parameters.

Now consider the bounding procedure for layer j. The linear layer from post-activation layer $j-1$ to pre-activation layer j can be encoded as:

$$\mathbf{W}^{(j)} a^{(j-1)}(x) + \mathbf{b}^{(j)} \le h^{(j)}(x) \le \mathbf{W}^{(j)} a^{(j-1)}(x) + \mathbf{b}^{(j)} \tag{36}$$

Consider the post activation function $a^{(j-1)}(x)$ of the unstable neuron $z_k^{(j-1)}$, before splitting we have $\underline{c} h_k^{(j-1)}(x) + \underline{d} \le a_k^{(j-1)}(x) \le \overline{c} h_k^{(j-1)}(x) + \overline{d}$, for some scalars $\underline{c}, \underline{d}, \overline{c}, \overline{d}$. After splitting, we now have $a_k^{(j-1)}(x) = h_k^{(j-1)}(x)$ for $\mathcal{C}_{sub}^l$ where $\underline{c} = \overline{c} = 1$, $\underline{d} = \overline{d} = 0$, since the unstable neuron is fixed to be active. By substituting the linear relaxation bounds before and after splitting in Equation 32, we obtain the bounding functions with regard to $h^{(j-1)}(x)$ in the following form:

$$\underline{\mathbf{A}}_j^{(j)} h^{(j-1)}(x) + \underline{\mathbf{b}}_j^{(j)} \le h^{(j)}(x) \le \overline{\mathbf{A}}_j^{(j)} h^{(j-1)}(x) + \overline{\mathbf{b}}_j^{(j)} \tag{37}$$

$$\underline{\mathbf{A}}_j^{(l,j)} h^{(j-1)}(x) + \underline{\mathbf{b}}_j^{(l,j)} \le h^{(j)}(x) \le \overline{\mathbf{A}}_j^{(l,j)} h^{(j-1)}(x) + \overline{\mathbf{b}}_j^{(l,j)} \tag{38}$$

By the fact the relaxation is fixed to be exact for $a_k^{(j-1)}(x)$, it holds that $\underline{\mathbf{A}}_j^{(j)} h^{(j-1)}(x) + \underline{\mathbf{b}}_j^{(j)} \le \underline{\mathbf{A}}_j^{(l,j)} h^{(j-1)}(x) + \underline{\mathbf{b}}_j^{(l,j)} \le \overline{\mathbf{A}}_j^{(l,j)} h^{(j-1)}(x) + \overline{\mathbf{b}}_j^{(l,j)} \le \overline{\mathbf{A}}_j^{(j)} h^{(j-1)}(x) + \overline{\mathbf{b}}_j^{(j)}$ for $\mathcal{C}_{sub}^l$.

Finally, for the bound propagation procedure of layer L, substituting the tightened bounding for $h^{(j-1)}(x)$, we obtain that $\underline{g_{l,i}}(x) = \underline{\mathbf{A}}^{(l,L)} x + \underline{\mathbf{b}}^{(l,L)} \ge \underline{\mathbf{A}}^{(L)} x + \underline{\mathbf{b}}^{(L)} = \underline{g_i}(x)$. ■

Corollary 10 *In each refinement iteration, the volume of the polytope under-approximation $\mathcal{T}_{Dom}$ does not decrease.*

Proof In each iteration of Algorithm 2, we replace the polytope $T_{\mathcal{C}_{sub}}(O)$ in a leaf subregion with two polytopes $T_{\mathcal{C}_{sub}^l}(O), T_{\mathcal{C}_{sub}^r}(O)$ in the DUP under-approximation. By Proposition 9, the total volume of the two new polytopes is at least that of the removed polytope. Thus the volume of the DUP approximation does not decrease.

Similarly, for ReLU splitting, we replace the polytope $T_{\mathcal{C}_{sub}}(O)$ in a leaf subregion with two polytopes $T_{\mathcal{C}^l_{sub}}(O), T_{\mathcal{C}^r_{sub}}(O)$ where the relaxed bounding functions for one unstable neuron are replaced with exact linear functions, i.e., $\underline{c}h_j^{(i)}(x) + \underline{d} \leq a_j^{(i)}(x) \leq \overline{c}h_j^{(i)}(x) + \overline{d}$ is replaced with the exact linear function $a_j^{(i)}(x) = h_j^{(i)}(x)$ and $a_j^{(i)}(x) = 0$, respectively, as shown in Figure 1 (from unstable to stable). By Proposition 9, the total volume of the two new polytopes is at least that of the removed polytope. Thus the volume of the DUP approximation does not decrease. ■

Proposition 7 *Algorithm 3 is sound for quantitative verification with input splitting.*

Proof Algorithm 3 outputs True only if, at some iteration, we have that the exact volume $\text{vol}(\mathcal{T}) \geq p \times \text{vol}(I)$. Since $\mathcal{T}$ is an under-approximation to the restricted preimage $f_I^{-1}(O)$, we have that $\frac{\text{vol}(f_I^{-1}(O))}{\text{vol}(I)} \geq \frac{\text{vol}(\mathcal{T})}{\text{vol}(I)} \geq p$, i.e. the quantitative property (I, O, p) holds. ■

Proposition 8 *Algorithm 3 is sound and complete for quantitative verification on piecewise linear neural networks with ReLU splitting.*

Proof The proof for the soundness of Algorithm 3 with ReLU splitting is similar to input splitting. Regarding the completeness, when all unstable neurons are fixed with one activation status, for each subregion $\mathcal{C}_{sub}$, we have $\underline{g_i}(x) = g_i(x)$. It then holds that for any $\mathcal{C}_{sub} \subset \mathcal{C}$ where $\bigcup \mathcal{C}_{sub} = \mathcal{C}$, $(\underline{g_i}(x) \geq 0) \wedge x \in \mathcal{C}_{sub} \iff (g_i(x) \geq 0) \wedge x \in \mathcal{C}_{sub}$, i.e., the polytope is the exact preimage. Hence, when all unstable neurons are fixed to an activation status, we have $\mathcal{T} = f_I^{-1}(O)$. Algorithm 3 returns False only if the volume of the exact preimage $\frac{\text{vol}(f_I^{-1}(O))}{\text{vol}(I)} = \frac{\text{vol}(\mathcal{T})}{\text{vol}(I)} < p$.

■

References

Aws Albarghouthi and Kenneth L. McMillan. Beautiful interpolants. In *Computer Aided Verification - 25th International Conference, CAV 2013, Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 2013. doi: 10.1007/978-3-642-39799-8_22.

David Avis and Komei Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, pages 98–104, 1991.

Daniel Ayala, Ouri Wolfson, Bo Xu, Bhaskar DasGupta, and Jie Lin. Parking slot assignment games. In *19th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS, Proceedings*, pages 299–308. ACM, 2011.

Teodora Baluta, Zheng Leong Chua, Kuldeep S. Meel, and Prateek Saxena. Scalable quantitative verification for deep neural networks. In *Proceedings of the 43rd International Conference on Software Engineering: Companion Proceedings*, ICSE '21, page 248–249. IEEE Press, 2021.

C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, pages 469–483, 1996. doi: 10.1145/235815.235821.

Patricia Mary Benoy. *Polyhedral domains for abstract interpretation in logic programming.* PhD thesis, University of Kent, UK, 2002.

Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *European Conference on Machine Learning and Knowledge Discovery in Databases*, volume 8190 of *Lecture Notes in Computer Science*, pages 387–402. Springer, 2013. doi: 10.1007/978-3-642-40994-3_25.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Rémy Boutonnet and Nicolas Halbwachs. Disjunctive relational abstract interpretation for interprocedural program analysis. In *Verification, Model Checking, and Abstract Interpretation - 20th International Conference, VMCAI 2019, Proceedings*, volume 11388 of *Lecture Notes in Computer Science*, pages 136–159. Springer, 2019. doi: 10.1007/978-3-030-11245-5_7.

Christopher Brix and Zhouxing Shi. Benchmarks for the vnn comp 2022. `https://github.com/VNN-COMP/vnncomp2022_benchmarks`, 2022. Accessed: 2022-09-30.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, 2016. URL `http://arxiv.org/abs/1606.01540`.

Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and Pawan Kumar Mudigonda. A unified view of piecewise linear neural network verification. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS*, pages 4795–4804, 2018.

Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and M Pawan Kumar. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, pages 1–39, 2020.

Augustin Chevallier, Frédéric Cazals, and Paul Fearnhead. Efficient computation of the the volume of a polytope in high-dimensions using piecewise deterministic markov processes. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2022, 28-30 March 2022, Virtual Event*, volume 151 of *Proceedings of Machine Learning Research*, pages 10146–10160. PMLR, 2022.

Felipe Codevilla, Matthias Müller, Antonio M. López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *Proceedings of the 2018 IEEE International Conference on Robotics and Automation*, pages 1–9, Brisbane, Australia, 2018. IEEE. doi: 10.1109/ICRA.2018.8460487.

Sumanth Dathathri, Sicun Gao, and Richard M. Murray. Inverse abstraction of neural networks using symbolic interpolation. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*, pages 3437–3444. AAAI Press, 2019. doi: 10.1609/aaai.v33i01.33013437.

Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. An abstraction-based framework for neural network verification. In *Computer Aided Verification: 32nd International Conference, CAV 2020, Proceedings, Part I 32*, pages 43–65. Springer, 2020.

Claudio Ferrari, Mark Niklas Müller, Nikola Jovanovic, and Martin T. Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 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 *2018 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2018.

Divya Gopinath, Hayes Converse, Corina S. Păsăreanu, and Ankur Taly. Property inference for deep neural networks. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*, ASE '19, page 797–809. IEEE Press, 2020. ISBN 9781728125084. doi: 10.1109/ASE.2019.00079.

Branko Grünbaum, Volker Kaibel, Victor Klee, and Günter M. Ziegler. *Convex polytopes*. Springer, New York, 2003. ISBN 9780387004242.

Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *Computer Aided Verification - 29th International Conference, CAV*

2017, Proceedings, Part I, volume 10426 of *Lecture Notes in Computer Science*, pages 3–29. Springer, 2017. doi: 10.1007/978-3-319-63387-9_1.

Kyle D. Julian and Mykel J. Kochenderfer. A reachability method for verifying dynamical systems with deep neural network controllers. *CoRR*, 2019. URL `http://arxiv.org/abs/1903.00520`.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Proceedings, Part I 30*, pages 97–117. Springer, 2017.

Matthias König, Xiyue Zhang, Holger H. Hoos, Marta Kwiatkowska, and Jan N. van Rijn. Automated design of linear bounding functions for sigmoidal nonlinearities in neural networks. *CoRR*, abs/2406.10154, 2024. doi: 10.48550/ARXIV.2406.10154.

Suhas Kotha, Christopher Brix, Zico Kolter, Krishnamurthy Dvijotham, and Huan Zhang. Provably bounding neural network preimages. *Accepted to NeurIPS 2023, CoRR*, 2023. doi: 10.48550/arXiv.2302.01404.

Suhas Kotha, Christopher Brix, Zico Kolter, Krishnamurthy Dvijotham, and Huan Zhang. INVPROP for provably bounding neural network preimages. `https://github.com/kothasuhas/verify-input`, accessed October, 2023.

Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, Mykel J Kochenderfer, et al. Algorithms for verifying deep neural networks. *Foundations and Trends in Optimization*, pages 244–404, 2021.

Ravi Mangal, Aditya V. Nori, and Alessandro Orso. Robustness of neural networks: a probabilistic and practical approach. In Anita Sarma and Leonardo Murta, editors, *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results, ICSE (NIER) 2019*, pages 93–96. IEEE / ACM, 2019.

Kyle Matoba and Francois Fleuret. Exact preimages of neural network aircraft collision avoidance systems. In *Proceedings of the Machine Learning for Engineering Modeling, Simulation, and Design Workshop at Neural Information Processing Systems 2020*, 2020.

Mark Niklas Müller, Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T Johnson. The third international verification of neural networks competition (vnn-comp 2022): Summary and results. *arXiv preprint arXiv:2212.10376*, 2022.

Pavithra Prabhakar and Zahra Rahimi Afzal. Abstraction based output range analysis for neural networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*, pages 15762–15772, 2019.

Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *Computer Aided Verification: 22nd International Conference, CAV 2010, Proceedings 22*, pages 243–257. Springer, 2010.

Nicholas Rober, Michael Everett, and Jonathan P How. Backward reachability analysis for neural feedback loops. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 2897–2904. IEEE, 2022.

Nicholas Rober, Sydney M Katz, Chelsea Sidrane, Esen Yel, Michael Everett, Mykel J Kochenderfer, and Jonathan P How. Backward reachability analysis of neural feedback loops: Techniques for linear and nonlinear systems. *IEEE Open Journal of Control Systems*, 2:108–124, 2023.

Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018*, pages 2651–2659. ijcai.org, 2018.

Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*, pages 9832–9842, 2019.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, pages 1–30, 2019.

Matthew Sotoudeh and Aditya V Thakur. Syrenn: A tool for analyzing deep neural networks. In *Tools and Algorithms for the Construction and Analysis of Systems: 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Proceedings, Part II 27*, pages 281–302. Springer, 2021.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Karim Tit, Teddy Furon, and Mathias Rousset. Efficient statistical assessment of neural network corruption robustness. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 9253–9263, 2021.

Vincent Tjeng, Kai Yuanqing Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *7th International Conference on Learning Representations, ICLR 2019*. OpenReview.net, 2019.

Joseph A Vincent and Mac Schwager. Reachable polyhedral marching (rpm): A safety verification algorithm for robotic systems with deep neural network components. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9029–9035. IEEE, 2021.

Benjie Wang, Stefan Webb, and Tom Rainforth. Statistically robust neural network classification. In *Uncertainty in Artificial Intelligence*, pages 1735–1745. PMLR, 2021a.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 29909–29921, 2021b.

Stefan Webb, Tom Rainforth, Yee Whye Teh, and M. Pawan Kumar. A statistical approach to assessing neural network robustness. In *7th International Conference on Learning Representations, ICLR 2019*. OpenReview.net, 2019a. URL `https://openreview.net/forum?id=S1xcx3C5FX`.

Stefan Webb, Tom Rainforth, Yee Whye Teh, and M. Pawan Kumar. A statistical approach to assessing neural network robustness. In *7th International Conference on Learning Representations, ICLR 2019*. OpenReview.net, 2019b.

Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International conference on machine learning*, pages 5286–5295. PMLR, 2018.

Weiming Xiang, Hoang-Dung Tran, Xiaodong Yang, and Taylor T Johnson. Reachable set estimation for neural network control systems: A simulation-guided approach. *IEEE Transactions on Neural Networks and Learning Systems*, 32(5):1821–1830, 2020.

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 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 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 *9th International Conference on Learning Representations, ICLR 2021, Virtual Event*. OpenReview.net, 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 *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Proceedings, Part I*, volume 12651 of *Lecture Notes in Computer Science*, pages 389–408. Springer, 2021.

Sangdoo Yun, Jongwon Choi, Youngjoon Yoo, Kimin Yun, and Jin Young Choi. Action-decision networks for visual tracking with deep reinforcement learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, pages 1349–1358. IEEE Computer Society, 2017.

Hang Zhang, Yuhao Zhang, and Xiangru Xu. Hybrid zonotope-based backward reachability analysis for neural feedback systems with nonlinear plant models. In *2024 American Control Conference (ACC)*, pages 4155–4161. IEEE, 2024a.

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 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018*, pages 4944–4953, 2018.

Xiyue Zhang, Benjie Wang, and Marta Kwiatkowska. Provable preimage under-approximation for neural networks. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 3–23. Springer, 2024b.

Xiyue Zhang, Benjie Wang, Marta Kwiatkowska, and Huan Zhang. PREMAP: A unifying preimage approximation framework for neural networks. `https://github.com/Zhang-Xiyue/PreimageApproxForNNs`, 2025. Accessed: 2025-06-11.

Yuhao Zhang, Hang Zhang, and Xiangru Xu. Backward reachability analysis of neural feedback systems using hybrid zonotopes. *IEEE Control Systems Letters*, 7:2779–2784, 2023.