PRIMA: Precise and General Neural Network Certification via Multi-Neuron Convex Relaxations

Mark Niklas Müller*‡, Gleb Makarchuk*‡, Gagandeep Singh†, Markus Püschel*, Martin Vechev*

*Department of Computer Science
ETH Zurich
Zurich, Switzerland
{mark.mueller, pueschel, martin.vechev}@inf.ethz.ch
gleb.makarchuk@gmail.com

†VMware Research and UIUC
United States
gagandeepsi@vmware.com,
ggnds@illinois.edu

***Abstract*—Formal verification of neural networks is critical for their safe and secure adoption in real-world applications. However, designing a precise and scalable verifier which can handle different activation functions, realistic network architectures and relevant specifications remains an open and difficult challenge.**

In this paper, we take a major step in addressing this challenge and present a new verification framework, called PRIMA. PRIMA is both (i) general: it handles any non-linear activation function, and (ii) precise: it computes precise convex approximations involving *multiple* neurons via novel convex hull approximation algorithms that leverage concepts from computational geometry. The algorithms have polynomial complexity, yield fewer constraints, and minimize precision loss.

We evaluate the effectiveness of PRIMA on a variety of challenging image classifiers from prior work. Our results show that PRIMA is significantly more precise than state-of-the-art, verifying robustness for up to 14%, 30%, and 34% more images than existing work on ReLU-, Sigmoid-, and Tanh-based networks, respectively. Further, PRIMA enables, for the first time, precise verification of a realistic neural network for autonomous driving within a few minutes.

***Index Terms*—Robustness, Certification, Convexity**

I. Introduction

The growing adoption of neural networks (NNs) in many critical settings raises the importance of ensuring they work safely and robustly when deployed in the real world [1]. While the last few years have seen significant progress in formal verification of NNs, existing deterministic methods either do not scale or are too imprecise when handling realistic networks.

Key challenge: Handling non-linearities. Computations in neural networks involve the application of non-linear activations in a layerwise manner. These functions can be applied many thousands of times leading to highly non-linear output regions. The main challenge in neural network verification then is in designing methods that can handle the effect of these non-linear functions in a precise and scalable manner.

Exact verifiers [2–16] are typically restricted to piecewise linear activations and do not scale to larger networks due to their exponential time complexity. To overcome this, incomplete verifiers [17–31] over-approximate the effects of non-linear functions by designing suitable convex relaxations. Most of these verifiers [17–29] are fundamentally based on single neuron convex approximations, i.e., activations are approximated separately. This approach leads to significant imprecision, as dependencies between neurons in the same layer are ignored. As a result, the approximation error can grow exponentially with each layer, leading to verification results which are too imprecise for proving the property of interest. To mitigate this limitation for ReLU networks, recent works either recursively split up the problem into multiple easier instances in what is called the Branch-and-Bound (BaB) approach [15, 16] or consider multi-neuron approximations that capture neuron interdependencies [30, 31]. Singh et al. [30] group neurons of a layer into small subsets of size $k > 1$ and then compute convex hulls *jointly* approximating the output of k ReLUs. Tjandraatmadja et al. [31] merge the activation layer with the preceding affine layer and compute a convex approximation over the resulting multivariate activation. These approaches yield state-of-the-art precision but are limited to ReLU activations and lack scalability. The multi-neuron relaxation approaches require small instances of the NP-hard convex hull problem to be solved exactly or large instances to be solved partially, while the BaB based methods have to consider increasingly more splits, which carries exponential cost.

This work: Precise multi-neuron approximations. In this work, we push the boundaries of the state-of-the-art in precise and scalable neural network verification and present a new general verification framework for networks with arbitrary, bounded, multivariate activation functions called PRIMA (PRecIse Multi-neuron Abstraction). PRIMA is based on a novel, general method, we call Partial Double Description Method (PDDM), for the precise and fast approximation of the convex hull problem for polytopes and can be applied to arbitrary specifications expressible as polyhedra such as individual fairness [32], global safety properties [8], acoustic [33], geometric [34], spatial [35], and ℓ_p-norm bounded perturbations [36]. Using PDDM as a subroutine, PRIMA uses our novel Split-Bound-Lift Method (SBLM) for the convex approximation of non-linear activation layers.

‡ Equal contribution.

Our experimental evaluation shows that PRIMA achieves state-of-the-art precision on the majority of our ReLU-based classifiers while being competitive on the rest. For Sigmoid- and Tanh-based networks, PRIMA significantly outperforms prior work on all benchmarks. Further, PRIMA enables, for the first time, the precise and scalable verification of a realistic architecture for autonomous driving containing $>$ 100k neurons in a regression setting. Finally, while PRIMA is incomplete, it can be used for boosting the scalability of state-of-the-art complete verifiers [13–16] for ReLU-based networks that benefit from precise convex approximations.

Main contributions. Our key contributions are:

1) The PDDM method for sound and precise approximation of the convex hull computation for polytopes, with worst-case polynomial time- and space-complexity and exactness guarantees in low dimensions.
2) The Split-Bound-Lift Method which can efficiently compute joint constraints over groups of non-linear functions, by decomposing the underlying convex hull problem into lower-dimensional spaces.
3) PRIMA, a novel verification framework combining these approaches with a sparse neuron grouping technique, to obtain the first multi-neuron abstraction framework for arbitrary, bounded, multivariate non-linear activation functions such as ReLU, Sigmoid, Tanh, and MaxPool.
4) We experimentally evaluate PRIMA on a range of ReLU-, Sigmoid-, and Tanh-based fully connected, convolutional, and residual networks and show that it is significantly more precise than the state-of-the-art, improving precision by up to 14%, 30%, and 34% for ReLU-, Sigmoid-, and Tanh-based networks, while also allowing the use in a regression setting and scaling to large networks, thus enabling applications of verification to real-world applications such as autonomous driving.
5) We release our code as part of the open-source ERAN framework at https://github.com/eth-sri/eran.

II. PROBLEM STATEMENT

In this section, we first establish the terminology we use to discuss neural networks (NNs), the notion of robustness and how it can be certified. Then, we explain the main challenge in NN robustness certification and the one addressed by our work: the precise and scalable handling of non-linear activation layers.

Notation. We use lower case Latin or Greek letters $a, b, x, \ldots, \lambda, \ldots$ for scalars, bold for vectors $\boldsymbol{a}$, capitalized bold for matrices $\boldsymbol{A}$, and calligraphic $\mathcal{A}$ or blackboard bold $\mathbb{A}$ for sets. Similarly, we denote scalar functions as $f : \mathbb{R}^d \to \mathbb{R}$ and vector valued functions bold as $\boldsymbol{f} : \mathbb{R}^{d \cdot k} \to \mathbb{R}^k$.

Neural networks. We focus our discussion on networks $\boldsymbol{h}(\boldsymbol{x}) : \mathcal{X} \to \mathbb{R}^{|\mathcal{Y}|}$ in the computer vision domain that map input samples (images) $\boldsymbol{x} \in \mathcal{X}$ to outputs $\boldsymbol{y} \in \mathbb{R}^{|\mathcal{Y}|}$. While our methods are applicable to arbitrary neural architectures [27], for simplicity we assume a feedforward architecture which is the interleaved composition of affine functions $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$, such as normalization, linear, convolutional, or average pooling layers, with non-linear activation layers $\boldsymbol{f}(\boldsymbol{x})$, composed from ReLU, Tanh, Sigmoid, or MaxPool:

$$\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{g}_L \circ \boldsymbol{f}_L \circ \boldsymbol{g}_{L-1} \circ ... \circ \boldsymbol{f}_1 \circ \boldsymbol{g}_0(\boldsymbol{x})$$

For a classification task, the network $\boldsymbol{h}$ classifies an input sample $\boldsymbol{x}$ by taking the argmax of its output: $c = \arg\max_j h(\boldsymbol{x})_j$.

Adversarial robustness. A classifier $\boldsymbol{h}$ is adversarially robust if the classification of an input $\boldsymbol{x}$ remains unchanged under small perturbations of the input. Formally, it means that $\boldsymbol{h}$ classifies all inputs in the p-norm ball

$$\mathbb{B}^p_\epsilon(\boldsymbol{x}) = \{\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{\eta} \mid |\boldsymbol{\eta}|_p \leq \epsilon\}$$

of radius ϵ to the same correct class:

$$\arg\max_j h(\boldsymbol{x})_j = \arg\max_j h(\boldsymbol{x}')_j, \quad \text{for all } \boldsymbol{x}' \in \mathbb{B}^p_\epsilon(\boldsymbol{x}). \quad (1)$$

The parameter ϵ bounds the admissible perturbations $\boldsymbol{\eta}$. For regression tasks, the network output is lower- and upper-bounded instead. We note that our methods are also applicable for verifying other properties beyond robustness such as fairness [32] and safety [8].

Neural Network Verification. Exact verification [8] has exponential complexity in the worst-case due to the thousands of non-linear activations $f : \mathbb{R} \to \mathbb{R}$ that either lead to a combinatorial blow-up of case distinctions (as for ReLU) or complicated shapes (as for Sigmoids). Therefore, state-of-the-art verifiers (e.g., [19, 24]) often sacrifice completeness for scalability by soundly over-approximating (defined below) with convex polyhedra when propagating $\mathbb{B}^p_\epsilon(\boldsymbol{x})$ through the network. These over-approximations are then chained to find a superset of all possible network outputs given a set of inputs.

Sound approximation. Affine layers map between convex polyhedra and can thus be captured exactly. The challenge are the non-linear activations, which introduce errors due to the needed over-approximations. To derive robustness guarantees, these over-approximations have to be sound. We define soundness of an approximation of function $\boldsymbol{f} : \mathbb{R}^{d \cdot k} \to \mathbb{R}^k$ given a set of inputs $\mathcal{P}_{\text{in}} \subseteq \mathbb{R}^{d \cdot k}$, and the approximation of the outputs $\mathcal{P}_{\text{out}} \subseteq \mathbb{R}^k$ as the property that $\mathcal{P}_{\text{out}}$ covers all possible outputs for inputs from $\mathcal{P}_{\text{in}}$. More formally, we require that $\forall \boldsymbol{x} \in \mathcal{P}_{\text{in}}, \boldsymbol{f}(\boldsymbol{x}) \in \mathcal{P}_{\text{out}}$.

As an example, consider the single-neuron ReLU shown in Figure 1, which maps the input x to $y \max(0, x)$. If bounds for x are known, $l_x \leq x \leq u_x$ with $l_x < 0$ and $u_x > 0$, the ReLU can be over-approximated by its convex hull, i.e., the triangle shown. The approximation errors grow exponentially with the network depth and can render these methods ineffective for verifying deeper networks. The challenge here is the trade-off between precision and scalability, discussed next.

Optimal approximation. Given a layer of n neurons, each applying the scalar, univariate, non-linear activation function $f(x)$ and the most precise polyhedral approximation $\mathcal{P}$ of the inputs $\boldsymbol{x}$, the most precise convex approximation after applying each f is given by the convex hull of all input output vector pairs $\text{conv}(\{(\boldsymbol{x}, \boldsymbol{f}(\boldsymbol{x})) \,|\, \boldsymbol{x} \in \mathcal{P} \subseteq \mathbb{R}^n\})$. Computing this hull is intractable due to the exponential cost $\mathcal{O}(n_v \log(n_v) + n_v^{n_n})$

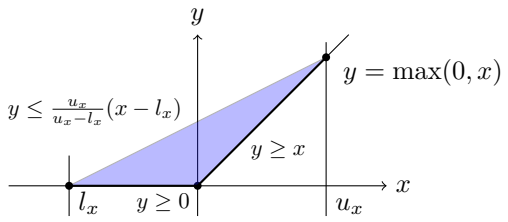


Figure 1: Single-neuron approximation of the ReLU $f(x) = \max(x, 0)$ with bounded inputs $x \in [l_x, u_x]$. The exact ReLU function (black) is nonconvex. A convex over-approximation is given by the three inequalities describing the blue triangle.

[37] in the number n_n of neurons, where the number of vertices $n_v = \mathcal{O}(n_c^{n_n})$ is at worst also exponential in the number of neurons [38], given the number of constraints n_c.

Neuron-wise approximation. For scalability reasons, almost all approximations of non-linear activations [19, 21, 24] operate separately on each neuron, as illustrated in Figure 1 for ReLU. They maintain upper and lower bounds l_x, u_x for each input x and compute convex hulls of all input-output tuples: $\text{conv}(\{(x, f(x)) \,|\, x \in [l_x, u_x] \subseteq \mathbb{R}\})$. The union of the obtained constraints is the final approximation of the layer. Geometrically, it is the Cartesian product of the convex hulls for each neuron, which is significantly larger (exponential in n) than the optimal convex hull discussed above (see Figure 4 later for a two-neuron example).

k-ReLU approximation. A first compromise between the optimal but intractable and imprecise but scalable neuron-wise approximation for the piecewise-linear ReLU activation was suggested in Singh et al. [30]. Their work computes convex hulls on smaller neuron groups of size k with simplified input bounds and combines the resulting constraints from many such groups to approximate the output of the entire layer. However, their approach only considers ReLU activations and relies on solving the convex hull problem exactly for every group in the $2k$-dimensional space. This heavily limits the number and size of groups they can process and therefore the approximation errors still grow substantially.

Our work. The fundamental challenge in pushing the limits of deterministic neural network verification is computing more precise and scalable convex approximations of activation layers. In this work, we address this challenge and introduce a general framework, called PRIMA, which can handle *all* common non-linear activation functions, including ReLU, Tanh, Sigmoid, and MaxPool. PRIMA works by combining a decompositional approach to reduce the dimensionality of the convex hull problems with a novel, fast, and precise approximate method for convex hull computations. This allows it to process up to two orders of magnitude more neuron groups, which yields much tighter approximations. Overall, PRIMA achieves significantly higher precision as well as speed-ups of up to an order of magnitude compared to convex-relaxation-based state-of-the-art methods [30, 31], while being competitive with highly optimized and fully GPU-based BaB methods [15] (which represent an orthogonal direction).

III. BACKGROUND ON POLYHEDRA

We now introduce the necessary background on polyhedra. In short, polyhedra can be represented using their extremal points ($\mathcal{V}$-representation) or using a set of linear constraints ($\mathcal{H}$-representation). Maintaining both representations of the same polyhedron simultaneously is called double description.

Vertex representation. A polyhedron $\mathcal{P} \subseteq \mathbb{R}^d$ is the closed convex hull of a set of generators called vertices $\mathcal{R} = \{x_i \in \mathbb{R}^d\}$:

$$\mathcal{P} = P(\mathcal{R}) = \left\{ \sum_i \lambda_i \boldsymbol{x}_i \,|\, \boldsymbol{x}_i \in \mathcal{R},\ \sum_i \lambda_i = 1,\ \lambda_i \in \mathbb{R}_0^+ \right\}.$$

A polyhedral cone $\mathcal{P} \subseteq \mathbb{R}^d$ is the positive linear span of a set of generators called rays $\mathcal{R} = \{x_i \in \mathbb{R}^d\}$:

$$\mathcal{P} = P(\mathcal{R}) = \left\{ \sum_i \lambda_i \boldsymbol{x}_i \,|\, \boldsymbol{x}_i \in \mathcal{R},\ \lambda_i \in \mathbb{R}_0^+ \right\}.$$

The origin is always included in a polyhedral cone. We call this representation of polyhedra vertex- or $\mathcal{V}$-representation.

Halfspace representation. Alternatively, a polyhedron can be described as the set $\mathcal{P} \subseteq \mathbb{R}^d$ satisfying a system of linear inequalities:

$$\mathcal{P} = \mathcal{P}(\boldsymbol{A}, \boldsymbol{b}) \equiv \{\boldsymbol{x} \in \mathbb{R}^d \,|\, \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}\} \tag{2}$$

with $\boldsymbol{A} \in \mathbb{R}^{m \times d}$ and $\boldsymbol{b} \in \mathbb{R}^m$. Geometrically, $\mathcal{P}$ is the intersection of m closed affine halfspaces $\mathcal{H}_i = \{x \in \mathbb{R}^d \mid \boldsymbol{a}_i \boldsymbol{x} \geq b_i\}$ with $\boldsymbol{a}_i \in \mathbb{R}^d$ and $b_i \in \mathbb{R}$. For a polyhedral cone $\boldsymbol{b} = \boldsymbol{0}$. We call this representation halfspace- or $\mathcal{H}$-representation. For convenience, a polyhedron $\mathcal{P}(\boldsymbol{A}, \boldsymbol{b})$ can be equivalently described in so-called homogenized coordinates $\boldsymbol{x'} = [1, \boldsymbol{x}]$, where it can be expressed as $\mathcal{P}(\boldsymbol{A'}) = \{\boldsymbol{x'} \in \mathbb{R}^{d+1} | \boldsymbol{A'}\boldsymbol{x'} \geq 0\}$ with the new constraint matrix $\boldsymbol{A'} = [-\boldsymbol{b}, \boldsymbol{A}]$.

We call a face of a d-dimensional polyhedron a vertex if it satisfies d linearly independent constraints[1] with equality and a facet if it satisfies exactly one linearly independent constraint with equality [39].

The rank of a ray or vertex in a d-dimensional polyhedron is the number of linearly independent constraints it satisfies with equality. We call a ray of rank $d-1$ and a vertex of rank d extremal. A ray of rank $d - n$ can be represented as the positive combination of n extremal rays and a vertex of rank $d - n$ as the convex combination of $n + 1$ extremal points.

Double description. Polyhedra analysis [40–42] usually maintains both $\mathcal{H}$- and $\mathcal{V}$-representation in a pair $(\boldsymbol{A'}, \mathcal{R})$, called double description. This is useful as computing the convex hull in the $\mathcal{V}$-representation is trivial (union of generator sets), but computing intersections is NP-hard. Conversely, computing intersections in $\mathcal{H}$-representation is trivial (union of constraints), but computing the convex hull is NP-hard. The transformation from the $\mathcal{V}$-representation to the $\mathcal{H}$-representation is called the *convex hull* problem and the reverse

[1] We call a set of constraints $\boldsymbol{a}_i \boldsymbol{x} \geq b_i$ linearly independent, if the $\boldsymbol{a}_i$ are linearly independent.

is called the *vertex enumeration* problem. Both are NP-hard in general and done on demand.

Inclusion. We define the inclusion of a polytope $\mathcal{Q}$ in a polytope $\mathcal{P}$ as: $\mathcal{Q} \subseteq \mathcal{P}$ or equivalently, $\forall \boldsymbol{x} \in \mathcal{Q}, \boldsymbol{x} \in \mathcal{P}$. In this setting, we say $\mathcal{P}$ over-approximates $\mathcal{Q}$ and $\mathcal{Q}$ under-approximates $\mathcal{P}$.

IV. Overview of PRIMA

We now present an overview of PRIMA, our framework for faster and more precise convex approximation of arbitrary, bounded, multivariate, non-linear activations. We explain our algorithm step-by-step including its key ideas. Full formalization is provided in subsequent sections.

We assume the setup as outlined in Section II: (1) an activation layer consisting of n neurons representing non-linear activations $f(x)$ (e.g., ReLU, Tanh, Sigmoid, ...), and (2) an n-dimensional polytope $\mathcal{S}$ constraining the input to the layer and providing neuron-wise bounds l_x, u_x.

PRIMA computes a convex over-approximation of the output using the following steps:

1) *Group decomposition:* Decompose the set of n neurons into overlapping groups (subsets) of size k.
2) *Octahedral projection:* For each group compute an octahedral over-approximation $\mathcal{P}$ of the projection of $\mathcal{S}$ to this group.
3) *Split-Bound-Lift Method (SBLM):* For each group of k neurons with input polytope $\mathcal{P}$, compute a convex over-approximation of the group output $\mathcal{K}$ in $\mathcal{H}$-representation using our novel SBLM method, by decomposing this problem to lower dimensions and leveraging our novel Partial Double Description Method (PDDM) with polynomial instead of exponential complexity to compute fast and scalable convex hull approximations.
4) *Combine constraints:* Finally, take the intersection of all obtained group-wise outputs $\mathcal{K}$ (union of their constraints) to obtain an over-approximation of the whole layer.

The input polytopes $\mathcal{S}$ are obtained using a fast, incomplete verifier (e.g., [19, 27, 28]) and the generated PRIMA constraints are added to an LP to yield the final verifier which we evaluate in Section VII. Our core contributions, SBLM and PDDM, in their synergy, are essential to make PRIMA tractable for non-linear activations and discussed in more detail later.

Next, we explain the basic workings of each step, identify key ideas, and illustrate the concepts in examples.

Group decomposition. Computing convex hulls for large sets of neurons (e.g. a whole layer) is infeasible. Thus, we restrict to groups of size k, typically $k = 3$–5. The key idea of the grouping is to capture dependencies between neurons ignored by neuron-wise approximations and thus achieve tighter approximations. The tightness increases with the number of groups and the degree of overlap between the groups. Considering all possible $\binom{n}{k}$ groups is too expensive; thus we define parameters n_s and s for tuning the cost and precision of approximations. We first partition the neurons of a layer into sets of size n_s and then for every set choose a subset of all $\binom{n_s}{k}$ groups that pairwise overlap by at most s, $1 \leq s < k$.

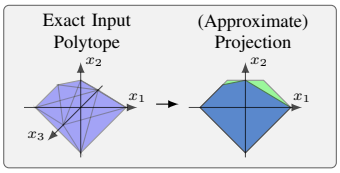


Figure 2: Exact projection of $\mathcal{S} \in \mathbb{R}^3$ (left) to $k = 2$ variables (blue) and its octahedral approximation $\mathcal{P}$ (green).

Octahedral projection. Projecting the input polytope $\mathcal{S}$ onto the input dimensions of every group is generally intractable due to the high dimensionality and large number of constraints. Therefore, we follow the idea of Singh et al. [30] and over-approximate the projection. Multidimensional octahedra [43], yielding $3^k - 1$ input constraints per group of k neurons, empirically provide a good trade-off between accuracy and complexity. This is illustrated for a layer of 3 neurons and $k = 2$ in Figure 2.

A. Split-Bound-Lift Method

The next and most demanding step takes a k-dimensional input polytope for a given k neuron group, and computes a $2k$-dimensional convex over-approximation of the output of the corresponding k activations. We design a new method called Split-Bound-Lift Method and illustrate its workings in Figure 3 on an example. We assume ReLU activations, group-size $k = 2$, and an octahedral input polytope $\mathcal{P}$ (left panel in Figure 3) described by:

$$\mathcal{P} = \{x_1 + x_2 \geq -2,\ -x_1 + x_2 \geq -2,\ x_1 - x_2 \geq -2,\\ -x_1 - x_2 \geq -2,\ -x_2 \geq -1.2\}.$$

Our method has three main steps:

Split the input polytope. We first split $\mathcal{P}$ into regions, which we call quadrants, for which tight or even exact, linear bounds of the activation functions are available. Choosing the right splits is essential for ensuring tight approximations. For our example with ReLU activation, the approximation error is minimized by splitting along the hyperplanes where the input variables x_1 and x_2 are 0. We chose the ordering $\{y_1, y_2\}$ of output variables and intersect $\mathcal{P}$ first with the halfspaces $\{\boldsymbol{x} \in \mathbb{R}^2 \,|\, x_1 \geq 0\}$ and $\{\boldsymbol{x} \in \mathbb{R}^2 \,|\, x_1 \leq 0\}$ and then $\{\boldsymbol{x} \in \mathbb{R}^2 \,|\, x_2 \geq 0\}$ and $\{\boldsymbol{x} \in \mathbb{R}^2 \,|\, x_2 \leq 0\}$ yielding the second and third column of polytopes in the central panel of Figure 3. For brevity we only follow the bottom path. There the two quadrants $\mathcal{P}_{2,1}$ and $\mathcal{P}_{2,2}$ are described by:

$$\mathcal{P}_{2,1} = \{x_1 - x_2 \geq -2,\ -x_1 \geq 0,\ -x_2 \geq -1.2,\ x_2 \geq 0\},$$
$$\mathcal{P}_{2,2} = \{x_1 + x_2 \geq -2,\ -x_1 \geq 0,\ -x_2 \geq 0\}.$$

Bound and lift the quadrants. In the second part of the algorithm, we lift these quadrants, one output variable at a time. As we will see later, this approach of extending quadrants only as needed enables significant gains in speed while reducing the approximation error. In our example, we first trivially

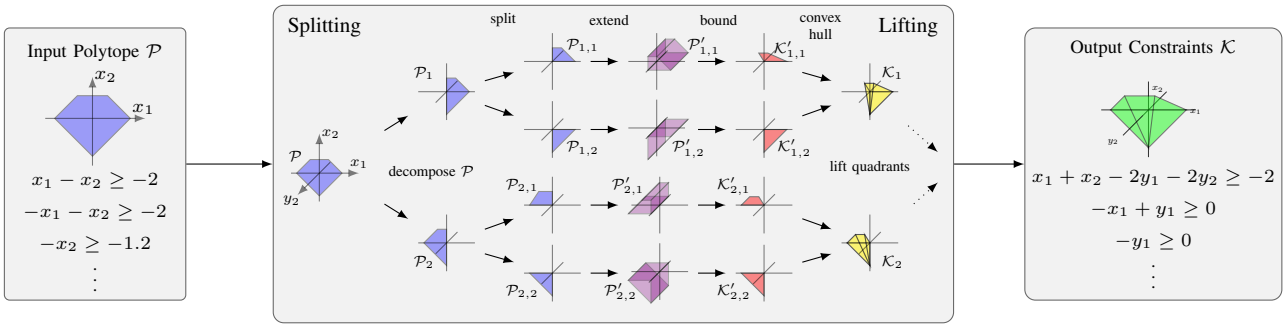


Figure 3: Illustration of the Split-Bound-Lift Method for a group of $k = 2$ neurons and a ReLU activation.

extend[2] all quadrants from the x_1-x_2 space into the y_2-x_1-x_2 space (purple column in Figure 3). Next we bound the added variable using the linear constraints corresponding to applying (a relaxation of) the activation function in the quadrant. Here $y_2 \leq 0$ and $y_2 \geq 0$ for the quadrant $\mathcal{P}_{2,2}$ (since $x_2 \leq 0$) and, $y_2 \leq x_2$ and $y_2 \geq x_2$ for the quadrant $\mathcal{P}_{2,1}$ (since $x_2 \geq 0$) and yielding the two polytopes (red column):

$$\mathcal{K}'_{2,1} = \{x_1 - x_2 \geq -2,\ -x_1 \geq 0,\ -x_2 \geq -1.2,\ x_2 \geq 0,\ x_2 - y_2 \geq 0,\ -x_2 + y_2 \geq 0\},$$
$$\mathcal{K}'_{2,2} = \{x_1 + x_2 \geq -2,\ -x_1 \geq 0,\ -x_2 \geq 0,\ -y_2 \geq 0,\ y_2 \geq 0\}.$$

Approximate convex hull. Next, we compute the convex hull of $\mathcal{K}'_{2,1}$ and $\mathcal{K}'_{2,2}$. Instead of using an exact method to compute the convex hulls, we utilize our PDDM to compute precise over-approximations, leveraging the concept of duality, ideas from computational geometry and our novel PDD polyhedron representation (explained below and in more detail in Section V). Note that this computation takes place in $3d$-space despite the group-output being in the $4d$ y_1-y_2-x_1-x_2-space.

This decomposition approach of extending quadrants only as needed, has two main effects: (i) directly computing $2k$-dimensional convex hulls with PDDM will lose more precision than our decomposed method, because PDDM is exact for polytopes of dimension up to 3 and loses precision only slowly for higher dimensions; (ii) a lower-dimensional polytope with fewer constraints, and generally also fewer vertices, significantly reduces the time required for the individual convex hull operations. Even our approximate method scales quartically ($\mathcal{O}\{n_a^4 \cdot n_v + n_a^2 \log(n_a^2)\}$) in the number of input constraints n_a and linear in the number of vertices n_v (see Appendix B for a proof of the complexity) while optimal exact methods are exponential ($\mathcal{O}(n_v \log(n_v) + n_v^{\lfloor d/2 \rfloor}$) [37]) in the number of dimensions and super linear in the number of input vertices. Note that for non-piecewise-linear functions (e.g. Tanh or Sigmoid), the number of vertices doubles for every extended dimension, making exact methods intractable and approximate methods not using the SBLM slow.

[2]Extending a d-dimensional polytope by a variable defines it in the $d+1$-dimensional space, where it is (initially) unbounded in the dimension of the added variable.

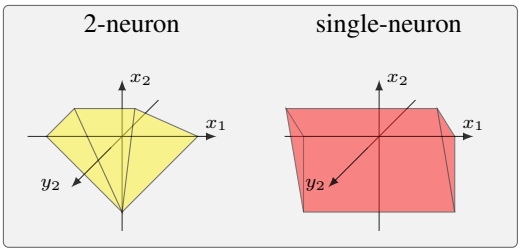


Figure 4: Comparison of 2-neuron and 1-neuron constraints on y_2 for a ReLU activation, given input polytope $\mathcal{P}$.

We now obtain the convex hull (yellow column) of the two polytopes $\mathcal{K}'_{2,1}$ and $\mathcal{K}'_{2,2}$ which is exact in our $3d$ case as:

$$\mathcal{K}_2 = \{x_1 + x_2 - 2y_2 \geq -2, -x_1 \geq 0,\ 0.375x_2 - y_2 \geq -0.75,\ -x_2 + y_2 \geq 0,\ y_2 \geq 0\}.$$

This already yields tighter bounds than the single neuron approximation (see Figure 4) and completes the first step of lifting. The next and in this case final step of lifting starts with extending $\mathcal{K}_2$, and the analogously computed $\mathcal{K}_1$, by y_1 into the y_1-y_2-x_1-x_2-space, where we apply bounds on y_1 yielding (in $4d$ and thus not illustrated):

$$\mathcal{K}'_{\mathbf{1}} = \{-x_1 + x_2 - 2y_2 \geq -2, x_1 \geq 0,\ 0.375x_2 - y_2 \geq -0.75,\ -x_2 + y_2 \geq 0,\ y_2 \geq 0,\ x_1 - y_1 \geq 0,\ -x_1 + y_1 \geq 0\},$$
$$\mathcal{K}'_{\mathbf{2}} = \{x_1 + x_2 - 2y_2 \geq -2, -x_1 \geq 0,\ 0.375x_2 - y_2 \geq -0.75,\ -x_2 + y_2 \geq 0,\ y_2 \geq 0,\ -y_1 \geq 0,\ y_1 \geq 0\}.$$

Completing the second step of lifting by computing their convex hull yields the tight 2-neuron constraints

$$\mathcal{K} = \{x_1 + x_2 - 2 \cdot y_1 - 2 \cdot y_2 \geq -2,\ 0.375 \cdot x_2 - y_2 \geq -0.75,\ -x_1 + y_1 \geq 0,\ -x_2 + y_2 \geq 0,\ y_1 \geq 0,\ y_2 \geq 0\}.$$

B. Partial Double Description Method (PDDM)

We develop the novel PDDM to compute *precise, fast and sound* over-approximations of convex hulls. This is in contrast to existing approximation methods, which either optimize for closer approximations [44–47] but sacrifice soundness, which is required for verification, or still have exponential complexity [48], making them too expensive for our application.

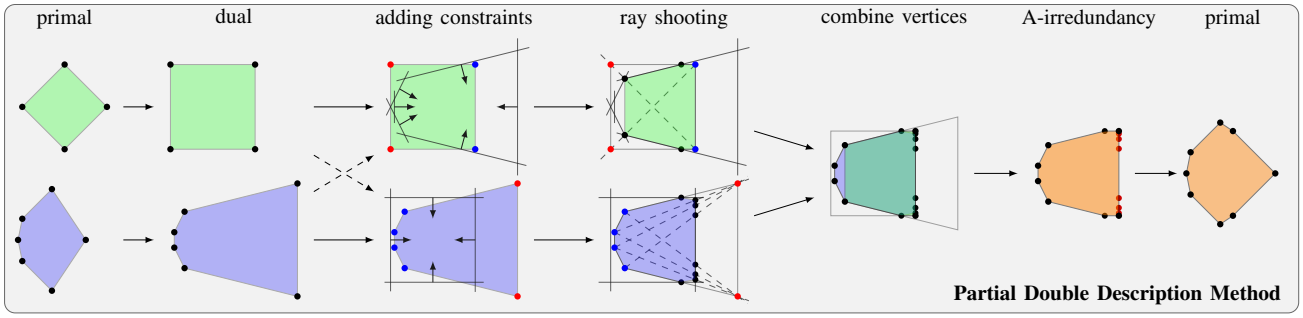


Figure 5: Illustration of the Partial Double Description Method for a 2-dimensional example. The input polytopes (1st column) are translated to their dual representation (2nd column), then all their constraints are added to the other dual polytope (3rd column). Now ray-shooting is used to discover vertices on the rays between points of the old polytope lying inside and outside the new constraints, by intersecting them with these constraints (4th column). These vertices are then combined (5th column) before A-irredundancy is enforced (6th column) and the result is translated back to primal space (7th column).

Double Description Method. The widespread Double Description Method (DDM) [40, 41] for computing the convex hull of two polyhedra first translates them to their dual representation, then intersects them in the dual space, by combining constraints one at a time, before translating the result back to primal space. Crucially, every step of adding an additional constraint generates new vertices quadratic in the number of input vertices, leading to an exponential increase.

Partial Double Description. We introduce the Partial Double Description (PDD) to guarantee soundness by design while allowing an approximate, much cheaper intersection in dual space. We combine an exact $\mathcal{H}$-representation, as their intersection is trivial, with an under-approximating[3] $\mathcal{V}$-representation, as their exact intersection carries exponential cost. Note that this definition does not yield a unique $\mathcal{V}$-representation for a given $\mathcal{H}$-representation or ensure a precise approximation; these properties are obtained by their construction through the PDDM.

Partial Double Description Method. Now, we define the PDDM to compute approximate convex hulls in PDD, leveraging two key insights: 1) Instead of adding the constraints of one polytope to the other one at a time during intersection (as per DDM), we add them all in a single step. Crucially, this leads to an overall number of vertices only quadratic (instead of exponential) in the number of original vertices. 2) This single-step approach is asymmetric and we can greatly increase accuracy, by performing it in both directions and combining the resulting vertices. Overall our approach yields a polynomial complexity (see Appendix B) for sound convex hull approximation (see Theorem B.1), guarantees exactness for low dimensions (see Theorem A.3), and empirically, is two orders of magnitude faster for the challenging cases in our experiments (see Figure 12c), and loses precision only slowly as dimensionality increases (see Figure 12b). We illustrate the Partial Double Description Method in Figure 5 and provide more technical details in Section V.

[3]An under-approximation in dual space corresponds to an over-approximation in primal space, due to inclusion reversion.

Combine constraints. In the last step, we combine the constraints forming the $\mathcal{H}$-representation of the computed output polyhedra of each group to obtain the final polyhedral over-approximation of the activation layer.

V. The Partial Double Description Method

In this section, we explain our PDDM for computing convex hull approximations in greater detail. First, we introduce the needed notion of duality and our novel Partial Double Description (PDD) representation for polyhedra. Then, we explain the PDDM step by step as illustrated in Figure 5.

PDDM computes the convex hull of two d-dimensional polytopes $\mathcal{P}_1 = \mathcal{P}(\boldsymbol{A}_1, \boldsymbol{b}_1)$ and $\mathcal{P}_2 = \mathcal{P}(\boldsymbol{A}_2, \boldsymbol{b}_2)$, but uses the equivalent homogenized representation (see Section III) of $(d+1)$-dimensional cones $\mathcal{P}'_1 = \mathcal{P}(\boldsymbol{A}'_1)$ and $\mathcal{P}'_2 = \mathcal{P}(\boldsymbol{A}'_2)$. Vertices in the original polytope now correspond to rays in the cone. In the following explanations we will use either term, depending on convenience.

The original polytope can be recovered from the cone, by intersecting it with the hyperplane $x'_0 = 1$ in primal, or with $x'_0 = -1$ in dual space (explained next) as visualized in Figure 6.

Duality. The dual $\overline{\mathcal{P}}$ of a polytope $\mathcal{P}$ with a minimal set (containing no redundancy) of extremal vertices $\mathcal{R}$ enclosing the origin but not containing it in its boundary (to ensure bounded dual) is defined as:

$$\overline{\mathcal{P}} = \{\boldsymbol{y} \in \mathbb{R}^d \mid \boldsymbol{x}^\top \boldsymbol{y} \leq 1 \; \forall \boldsymbol{x} \in \mathcal{P}\} \tag{3}$$

$$= \bigcap_{x \in \mathcal{R}} \{\boldsymbol{y} \in \mathbb{R}^d \mid \boldsymbol{x}^\top \boldsymbol{y} \leq 1\}. \tag{4}$$

and for polyhedral cones $\mathcal{P}'$ [49]:

$$\overline{\mathcal{P}'} = \{\boldsymbol{y}' \in \mathbb{R}^{d+1} \mid \boldsymbol{x}'^\top \boldsymbol{y}' \leq 0 \; \forall \boldsymbol{x}' \in \mathcal{P}'\}. \tag{5}$$

Figure 6 shows an example of the dual of a polytope. Important for the remaining section are three properties of the transform between primal and dual. (1) It is inclusion reversing: $\mathcal{P} \subset \mathcal{Q}$ if and only if $\overline{\mathcal{Q}} \supset \overline{\mathcal{P}}$. (2) The $\mathcal{V}$-representation of the dual corresponds to the $\mathcal{H}$-representation of the primal and vice

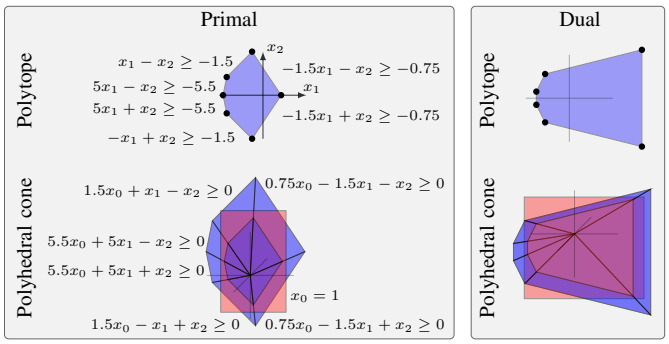
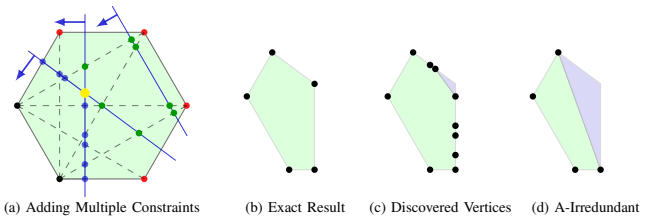


Figure 6: Top row: 2d-polytope in primal (left) and dual (right) space. Bottom row: equivalent polyhedral cones in homogenized coordinates. In red, we show the plane the polyhedral cone can be intersected with to recover the polytope.



(a) Adding Multiple Constraints (b) Exact Result (c) Discovered Vertices (d) A-Irredundant

Figure 7: Adding a batch of three constraints (blue thick lines) to a polytope in PDD. Vertices are separated into $\mathcal{R}'_+$ (black), $\mathcal{R}'_0$ (none), and $\mathcal{R}'_-$ (red). Ray-shooting discovers new vertices $\mathcal{R}'_*$ (blue), avoiding the superfluous green points, but missing an extremal vertex (yellow) (a). Exact intersection (b), result of joint constraint processing (c), and under-approximation after enforcing A-irredundancy (d).

versa: $\mathcal{P} = \mathcal{P}(\boldsymbol{A}', \mathcal{R}')$ implies $\overline{\mathcal{P}} = \mathcal{P}(\mathcal{R}'^{\top}, \boldsymbol{A}'^{\top})$, where $(\cdot)^{\top}$ denotes transpose. (3) The dual of the dual of a polyhedron is the original primal polyhedron $\overline{\overline{\mathcal{P}}} = \mathcal{P}$.

Partial Double Description. We leverage these duality properties in two ways: we translate the convex hull problem in primal space to an intersection problem in dual space and obtain an $\mathcal{H}$-representation *over-approximating* the convex hull in primal space by computing a $\mathcal{V}$-representation *under-approximating* the intersection in dual space. To do so efficiently, we introduce the Partial Double Description (PDD) as a relaxation of the Double Description (DD) (Section III) as discussed in the overview.

Formally, the PDD of a $(d+1)$-dimensional polyhedral cone is the pair $(\boldsymbol{A}', \mathcal{R}')$ with $\boldsymbol{A}' \in \mathbb{R}^{m\times(d+1)}$ and $\mathcal{R}' \in \mathbb{R}^{n\times(d+1)}$ where the $\mathcal{V}$-representation is an under-approximation of the $\mathcal{H}$-representation or more formally, where for any row $\boldsymbol{r} \in \mathcal{R}'$ and $\boldsymbol{a} \in \boldsymbol{A}'$, $\boldsymbol{a} \cdot \boldsymbol{r} \geq 0$ holds.

We call constraints $\boldsymbol{a}_j \in \boldsymbol{A}'$ *active* for a given ray $\boldsymbol{r}_i \in \mathcal{R}'$, if they are fulfilled with equality, that is $\boldsymbol{a}_j \boldsymbol{r}_i = 0$. We store this relationship as part of the PDD in what we call the incidence matrix $\mathcal{I} \in \{0,1\}^{n\times m}$: $\mathcal{I}_{i,j} = 1$ if $\boldsymbol{a}_j \boldsymbol{r}_i = 0$ and $\mathcal{I}_{i,j} = 0$ otherwise. Further, we define the row-wise inclusion relationship on $\mathcal{I}$: $\mathcal{I}_i \subseteq \mathcal{I}_j$ if $\mathcal{I}_{i,k} \leq \mathcal{I}_{j,k}$, $1 \leq k \leq m$.

Next, we describe PDDM as illustrated in Figure 5.

A. Conversion to Dual

Given an input polyhedral cone in PDD representation $(\boldsymbol{A}', \mathcal{R}')$ (1st column in Figure 5), the first step of the PDDM is to convert it to its dual space representation $(\mathcal{R}'^{\top}, \boldsymbol{A}'^{\top})$ [50] (2nd column in Figure 5).

B. Intersection

The next step in the PDDM is the intersection in dual space itself (columns 3 to 5 in Figure 5). The standard approach for the intersection of polyhedra in DD is to sequentially add the constraints of one polytope to the other, computing exact $\mathcal{V}$-representations at every step. This however can increase the number of vertices quadratically in every step resulting in an exponential size of the intermediate representation. Instead, we add all constraints jointly in one step, leveraging our PDD. In the following description, we adopt the polytope (not cone) view.

Batch intersection. To intersect a polytope $(\boldsymbol{A}', \mathcal{R}')$ in PDD with a batch of constraints represented by the matrix $\widetilde{\boldsymbol{A}}$, we separate the vertices in $\mathcal{R}'$ into three sets depending on whether they satisfy all new constraints with inequality ($\mathcal{R}'_+$), some only with equality ($\mathcal{R}'_0$), or violate at least one ($\mathcal{R}'_-$). An example is shown in Figure 7(a): the three constraints are shown in blue and the vertices as $\mathcal{R}'_+$ (black), $\mathcal{R}'_0$ (none), and $\mathcal{R}'_-$ (red).

Now we employ a technique called ray-shooting [51] and shoot a ray $\overrightarrow{\boldsymbol{r}_+\boldsymbol{r}_-}$ from a vertex $\boldsymbol{r}_+ \in \mathcal{R}'_+$ "inside" the intersection $\boldsymbol{A}' \cap \widetilde{\boldsymbol{A}}$ to a vertex $\boldsymbol{r}_- \in \mathcal{R}'_-$ "outside" the intersection. We record the first hyperplane $\mathcal{H} = \{\boldsymbol{x} \in \mathbb{R}^d \,|\, \widetilde{\boldsymbol{a}}_i \boldsymbol{x} = 0\}$ corresponding to one of the new constraints $\widetilde{\boldsymbol{a}}_i \in \widetilde{\boldsymbol{A}}$ that intersects with the ray $\overrightarrow{\boldsymbol{r}_+\boldsymbol{r}_-}$. We add the point $\boldsymbol{r}_*$ at which $\overrightarrow{\boldsymbol{r}_+\boldsymbol{r}_-}$ intersects $\mathcal{H}$ to the result. Doing so for all combinations of $(\boldsymbol{r}_+, \boldsymbol{r}_-) \in \mathcal{R}'_+ \times \mathcal{R}'_-$ yields the set of points

$$\mathcal{R}'_* = \{\boldsymbol{r}_* = \overrightarrow{\boldsymbol{r}_+\boldsymbol{r}_-} \cap \mathcal{H} \,|\, (\boldsymbol{r}_+, \boldsymbol{r}_-) \in \mathcal{R}'_+ \times \mathcal{R}'_-\}. \quad (6)$$

The $\mathcal{V}$-representation of the resulting intersection is now the union $\mathcal{R}'_+ \cup \mathcal{R}'_0 \cup \mathcal{R}'_*$. In Figure 7 (a) the rays $\overrightarrow{\boldsymbol{r}_+\boldsymbol{r}_-}$ are dashed lines from all black to all red vertices and discover new vertices $\mathcal{R}'_*$ (blue). Only using the first intersections, immediately discards the green points, however we also do not discover the yellow point, which is an extremal vertex of the exact intersection (b), obtaining instead the under-approximation (c).

Boosting precision. Batch intersection is asymmetric: The PDD of one polytope is intersected with the $\mathcal{H}$-representation of another, to obtain an exact $\mathcal{H}$-representation and under-approximating $\mathcal{V}$-representation of the intersection (compare Figure 8 (b) and (c)). We perform it in both directions, obtaining two under-approximations. Their convex hull (obtained by the union of vertices) is still an under-approximation of the exact intersection and more precise than the individual

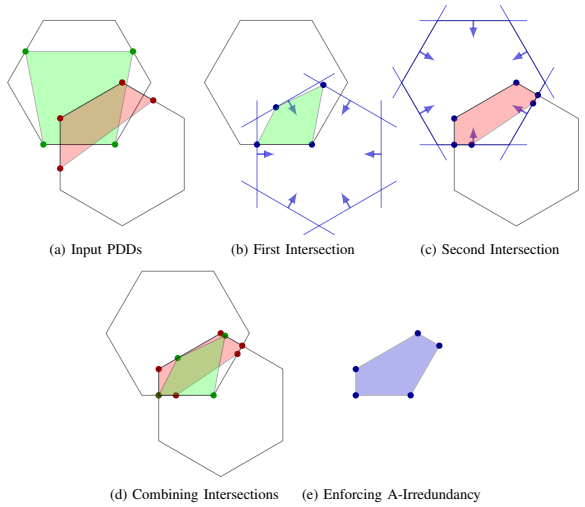


Figure 8: Boosting intersection precision by combining both directions of batch intersection. Input polytopes in PDD with exact $\mathcal{H}$-representation (black) and approximate $\mathcal{V}$-representation ($\mathcal{P}_1$ green and $\mathcal{P}_2$ red) (a), batch intersection of $\mathcal{P}_1$ with the $\mathcal{H}$-representation of $\mathcal{P}_2$ (b), batch intersection in the opposite direction (c), combining both intersections (d), and applying A-irredundancy (e).

under-approximations. This is illustrated in Figure 8, where the exact intersection (blue in (e)) of the two $\mathcal{H}$-representations (black in (a)) is recovered despite the union of the input $\mathcal{V}$-representations (green and red in (a)) not covering it. This is due to the synergy between PDD and PDDM: the under-approximate $\mathcal{V}$-representation of the first polytope is intersected with the exact $\mathcal{H}$-representation of the second one and vice versa. This is crucial to minimize the precision loss due to using approximations.

In our experimental results, this yields a significant boost in precision. Further, the intersection results are exact for small dimensions $d \leq 4$ of cones (see Appendix A for the proof).

C. Enforcing A-Irredundancy

Despite using batch intersection, the number of vertices can grow quickly when computing multiple convex hulls sequentially in the Split-Bound-Lift Method. Therefore, some notion of redundancy is needed to efficiently reduce the representation size. The standard definitions of irredundancy are: (1) the set of unique extremal rays of the cone $\mathcal{P}(\boldsymbol{A}')$ are irredundant, and (2) a ray $\boldsymbol{r}_i$ is irredundant if removing it leads to a different cone $\mathcal{P}(\mathcal{R}') \neq \mathcal{P}(\mathcal{R}' \setminus \boldsymbol{r}_i)$. For an exact DD, an irredundant representation does not lose precision and can be computed by retaining only rays with rank $d-1$. However, a PDD $(\boldsymbol{A}', \mathcal{R}')$ usually does not include all or even any extremal rays of the cone $\mathcal{P}(\boldsymbol{A}')$. Consequently, enforcing the first irredundancy definition could remove all rays, enforcing the second one is expensive to compute in the absence of a full set of extremal rays.

Therefore, we propose the concept of A-irredundancy to balance the cost and precision of our algorithm, which we define by requiring for all rays $\boldsymbol{r}_i \in \mathcal{R}'$ that there may not be another generator $\boldsymbol{r}_j \in \mathcal{R}'$ with a larger (by inclusion) active constraint set:

$$\mathcal{I}_i \not\subseteq \mathcal{I}_j, \quad \text{for all } i, j \in \{1, ..., n\},\ i \neq j.$$

Any ray fulfilling a subset (including the same) constraints with equality as another ray, is removed to obtain an A-irredundant representation. Extremal rays will always be retained as they have the maximum number of active constraints and there are never two with the same active set. Intuitively, this enforces that no two rays lie in the interior of the same face of the polyhedron.

We illustrate the effect of enforcing A-irredundancy in Figure 7 where we use it to obtain the polytope 7(d) from 7(c) and see that the resulting reduction in generator set size can come at the cost of precision loss. Enforcing A-irredundancy in the 6$^{\text{th}}$ column of Figure 5 (removing the red vertices), recovers the minimal set of extremal rays. Translating the resulting PDD back to primal space concludes the PDDM. See Appendix C for a full proof of the soundness of the PDDM.

VI. Split-Bound-Lift Method

In this section, we explain the Split-Bound-Lift Method in greater detail. Recall that we use the SBLM to compute k-neuron abstractions, by approximating the convex hull $\text{conv}(\{(\boldsymbol{x}, \boldsymbol{f}(\boldsymbol{x})) \,|\, \boldsymbol{x} \in \mathcal{P} \subseteq [l_x, u_x]^k\})$ for a group of k neurons and their activation functions $\boldsymbol{f}(\boldsymbol{x}) = [f_1(x_1), ..., f_k(x_k)]^\top$, assuming that their inputs are constrained by the polytope $\mathcal{P}$.

At a high level, we first decompose the input polytope into regions where we can bound all activation functions tightly. Then, we extend these regions into the output space and apply linear constraints corresponding to the (relaxed) activations. Taking the convex hull of the resulting polytopes yields an $\mathcal{H}$-representation encoding the k-neuron abstraction.

To increase the efficiency of this approach, we use a decomposition method we call *splitting* and then recursively extend and bound the resulting polytopes by one output variable at a time, which we call *lifting*, to minimize the dimensionality in which we have to compute the convex hulls. We formalize this in Algorithm 3 in Appendix D and explain both splitting and lifting below after stating the prerequisites for the SBLM.

A. Prerequisites

We assume just one type of activation function $f : \mathbb{D} \to \mathbb{R}$ with domain $\mathbb{D}$ is to be bounded. Now SBLM requires a set of intervals $\mathcal{D}^i$ (e.g. $x_j \leq 0, x_j \geq 0$ for ReLU), covering the domain $\mathbb{D}$ (e.g. $\mathbb{R}$ for ReLU), and tight linear constraints $\mathcal{B}^i$ on the function output (e.g. $y_j = 0, y_j = x_j$ for ReLU) on the interval obtained by intersecting the bounding box of $\mathcal{P}$ with $\mathcal{D}^i$. More formally, we require the intervals

$$\mathcal{D}^i = [c_i, d_i], \quad c_i, d_i \in \overline{\mathbb{R}} \text{ and } c_i \leq d_i$$
$$\mathbb{D} \subseteq \bigcup_i \mathcal{D}^i$$

with the affinely extended real numbers $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$ and the bounds on these intervals

$$\mathcal{B}^i = (a_i^{\leq}, a_i^{\geq}), \quad a_i^{\{\leq,\geq\}}(x) = ax + b,\ a, b \in \mathbb{R} \quad s.t.$$
$$a_i^{\leq}(x) \leq f(x) \leq a_i^{\geq}(x), \quad \forall\, x \in (\mathcal{D}^i \cap [l_x, u_x]_i)$$

to be provided to instantiate SBLM and by extension PRIMA. We note that the bounds c_i and d_i of the bounding regions can depend on the concrete input bounds l_x and u_x and the slope a and intercept b of $a_i^{\{\leq,\geq\}}$ can in turn depend on the corresponding concrete interval bounds $[\max(l_x, c_i), \min(u_x, d_i)]$.

Generalization. While we focus on the univariate case using only two bounding regions $\mathcal{D}^1$ and $\mathcal{D}^2$ in the following, SBLM and by extension PRIMA can be generalized to allow for neuron groups combining different multivariate activation functions $f : \mathbb{D} \subseteq \mathbb{R}^d \to \mathbb{R}$. Further, more than one upper- and lower-bound $\mathcal{B}^i$ per bounding region can be provided and $\mathcal{D}^i$ can be specified as a polyhedral region instead of as an interval, as long as their union covers the domain $\mathbb{D} \subseteq \bigcup_i \mathcal{D}^i$ of the individual functions f.

B. Splitting the Input Polytope

To apply the bounds $\mathcal{B}^i$, the input polytope $\mathcal{P}$ has to be split into the regions for which the bounds were specified. These regions correspond to the intersection of $\mathcal{P}$ with the k-cartesian product of the bounding regions $\mathcal{D}^i$. We choose an ordering of the output variables $\mathcal{I}$ and recursively split $\mathcal{P}$ by intersecting with the bounding regions associated with these output variables.

As every such split is equivalent, we will explain one case assuming the parent polytope $\mathcal{P}_1$, the output variable $y_j = f(x_j)$, and the corresponding bounding regions $\mathcal{D}_j^1 = \{\boldsymbol{x} \in \mathbb{R}^k \,|\, x_j \geq c_1\}$ and $\mathcal{D}_j^2 = \{\boldsymbol{x} \in \mathbb{R}^k \,|\, x_j \leq d_2\}$. We compute the children nodes by intersecting $\mathcal{P}_j$ with $\mathcal{D}_j^1$ and $\mathcal{D}_j^2$ to obtain $\mathcal{P}_{1,1} = \mathcal{P}_1 \cap \mathcal{D}_j^1$ and $\mathcal{P}_{1,2} = \mathcal{P}_1 \cap \mathcal{D}_j^2$. Starting with $\mathcal{P}$ at the root and recursively applying this splitting rule for every $y_j \in \mathcal{I}$, generates a polytope tree, which we call the decomposition tree, with 2^k leaf polytopes $\mathcal{P}_{j^k}$, which we call *quadrants*. This is illustrated in the blue portion of the central panel in Figure 3, where $\mathcal{D}^1$ and $\mathcal{D}^2$ are $\mathbb{R}_0^+$ and $\mathbb{R}_0^-$, respectively.

C. Bound & Lift

We now extend these quadrants to the output space and bound them using the corresponding bounds on the activation function, before taking their convex hull. This yields a polytope $\mathcal{K}$, jointly constraining the inputs and outputs of a neuron group. The constraints of its $\mathcal{H}$-representation form the desired k-neuron abstraction. We call this process *lifting* and propose a recursive approach: We lift sibling polytopes on the decomposition tree until only the desired polytope $\mathcal{K}$ remains.

Again, we explain a single step of lifting, as they are equivalent. We assume the sibling polytopes $\mathcal{K}_{1,1}$ and $\mathcal{K}_{1,2}$, corresponding to $\mathcal{P}_{1,1}$ and $\mathcal{P}_{1,2}$ in the decomposition tree, with the associated input- and output-variables x_j and y_j, respectively, and the pairs of bounds $\mathcal{B}_j^1$ and $\mathcal{B}_j^2$ instantiated for y_j. A single step involves:

- Extend $\mathcal{K}_{1,1}$ and $\mathcal{K}_{1,2}$ by the output variable y_j,
- Bound y_j on the extended polytopes, by intersecting them with the constraints $\mathcal{B}_j^1$ and $\mathcal{B}_j^2$ to obtain $\mathcal{K}'_{1,1}$ and $\mathcal{K}'_{1,2}$,
- Compute their convex hull using the PDDM: $\mathcal{K}_1 = \mathrm{conv}(\mathcal{K}'_{1,1}, \mathcal{K}'_{1,2})$

Applying this lifting rule recursively to the decomposition tree, combines all 2^k quadrants into a single $2k$-dimensional polytope $\mathcal{K}$, jointly constraining the inputs and outputs, thereby concluding the Split-Bound-Lift Method. This is illustrated in the right portion of the central panel in Figure 3. The decompositional approach has two benefits: First, computing approximate convex hulls via the PDDM is exact for polytopes of dimension up to 3 and starts to lose precision only slowly as dimensionality increases. Directly computing $2k$-dimensional convex hulls with PDDM will therefore lose more precision than using our decomposed method. Secondly, a lower-dimensional polytope with fewer constraints and generally also fewer vertices significantly reduces the runtime for the individual convex hull operations. In fact, computing the convex hulls for the approximation of non-piecewise linear functions directly in the input-output space is intractable even for groups of only size $k = 3$, as the number of vertices increases exponentially with k during bounding in that case.

D. Instantiation for various functions

We instantiate SBLM for common network functions next.

ReLU. We can capture all univariate, piecewise linear functions, such as ReLU exactly on the intervals $\mathcal{D}^i$ where they are linear. Further, if the neuron-wise bounds $[l_x, u_x]$ only contain one such linear region, the neuron behaves linearly, can be encoded exactly and is excluded from the k-neuron abstraction. Therefore, we consider $y = max(x, 0)$ with $x \in [l_x, u_x]$ for $l_x < 0 < u_x$. We choose $\mathcal{D}^1 = [-\infty, 0]$ and $\mathcal{D}^2 = [0, \infty]$, with $\mathcal{B}^1 = (y \geq 0,\ y \leq 0)$ and $\mathcal{B}^2 = (y \geq x,\ y \leq x)$.

Tanh and Sigmoid. Let f be an S-curve function with domain $[l_x, u_x]$, that is $f''(x) \geq 0$ for $x \leq 0$, $f''(x) \leq 0$ for $x \geq 0$ and $f'(x) > 0$ for $x \in [l_x, u_x]$. Both Sigmoid $\sigma(x) = \frac{e^x}{e^x+1}$ and Tanh $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ have these properties. We split the domain at $c \in [l_x, u_x]$ into $\mathcal{D}^1 = [-\infty, c]$ and $\mathcal{D}^2 = [c, \infty]$ to minimize the area between upper and lower bound in the input-output plane, via the bounds from Singh et al. [19]

$$x \leq f(u_d) + (x - u_d) \begin{cases} \frac{f(u_d) - f(l_d)}{u_d - l_d} & \text{if} \quad u_d \leq 0 \\ \min(f'(u_d), f'(l_d)) & \text{else} \end{cases}$$
$$x \geq\ f(l_d) + (x - l_d) \begin{cases} \frac{f(u_d) - f(l_d)}{u_d - l_d} & \text{if} \quad l_d \geq 0 \\ \min(f'(u_d), f'(l_d)) & \text{else,} \end{cases}$$

where we denote the lower bound of the intersection $\mathcal{D}^i \cap [l_x, u_x]$ as l_d and the upper one as u_d. We show these bounds in Figure 9 for the Sigmoid function and, for illustration purposes, a non-optimal c.

MaxPool. Let MaxPool be the multivariate function $y = \max(x_1, x_2, ..., x_d)$ on the domain $\boldsymbol{x} \in \mathcal{P} \subseteq [l_x, u_x]^d$. Note, that here the generalized formulation is required. We chose

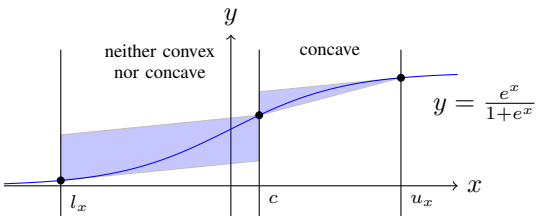


Figure 9: Interval-wise bounds for the Sigmoid function on the intervals $[l_x, c]$ and $[c, u_x]$.

the polyhedral bounding regions $\mathcal{D}^i = \{\boldsymbol{x} \in \mathbb{R}^d | x_i \geq x_j,\ 1 \leq j \leq d,\ i \neq j\}_i$, separating the domain into the d regions where one variable dominates all others. On each of these regions, MaxPool can be bounded exactly with $y \leq x_i$ and $y \geq x_i$. During the splitting process, this increased number of bounding regions leads to a decomposition tree where every parent node has d child nodes.

VII. Experimental Evaluation

In this section, we evaluate the effectiveness of PRIMA and show that it significantly improves over state-of-the-art verifiers on a range of challenging benchmarks yielding up to 14%, 30% and 34% precision gains on ReLU-, Sigmoid-, and Tanh-based networks, respectively. Further, we show that PRIMA can scale to obtain tight bounds on a real-world autonomous driving regression task. Finally, we demonstrate the effectiveness and benefits of computing relaxations with SBLM compared to directly using the exact convex hull.

A. Experimental setup

The neural network certification benchmarks for fully connected networks were run on a 20 core 2.20GHz Intel Xeon Silver 4114 CPU with 100 GB of main memory and those for convolutional networks on a 16 Core 3.6GHz Intel i9-9900K with 64GB of main memory and an NVIDIA RTX 2080Ti. We use Gurobi 9.0 for solving MILP and LP problems [52].

B. Benchmarks

We evaluate PRIMA on a wide range of networks based on ReLU, Tanh, and Sigmoid activations (see also Table I):

- The set of fully-connected and convolutional ReLU networks[4] from Singh et al. [30] trained using DiffAI [23], PGD [53], Wong [54], and natural training (see results on MNIST and CIFAR10 in Table II).
- The published set of CIFAR10 convolutional networks from Dathathri et al. [55], trained using either just PGD or a mix of normal and PGD training (see results on CIFAR10 in Table III).
- The set of fully-connected and convolutional Tanh and Sigmoid networks from Singh et al. [30] trained using natural training (see results on MNIST in Table IV).

[4]The networks referred to as $6 \times \cdot 00$ and $9 \times \cdot 00$ in previous work only include 5 and 8 hidden layers, respectively, and have therefore been renamed.

Table I: Neural network architectures used in experiments.

Dataset	Model	Type	Neurons	Layers	Activation
MNIST	5×100^4	FC	510	5	ReLU
	6×100	FC	600	6	Tanh/Sigm
	8×100^4	FC	810	8	ReLU
	9×100	FC	900	9	Tanh/Sigm
	5×200^4	FC	1 010	5	ReLU
	6×200	FC	1 200	6	Tanh/Sigm
	8×200^4	FC	1 610	8	ReLU
	`ConvSmall`	Conv	3 604	3	Tanh/Sigm
	`ConvBig`	Conv	34 688	6	ReLU
CIFAR10	`ConvSmall`	Conv	4 852	3	ReLU
	`CNN-A-Mix`	Conv	6 244	3	ReLU
	`CNN-B-Adv`	Conv	16 634	3	ReLU
	`ConvBig`	Conv	62 464	6	ReLU
	`ResNet`	Residual	107 496	13	ReLU
Self-Driving	`DAVE`	Conv	107 032	9	ReLU + Tanh

- The NVIDIA self-driving car network architecture `DAVE` [56] trained on a steering angle prediction task using the Udacity self-driving car dataset [57]. We use 31834 train and 1974 test samples[5], an input resolution of $3 \times 66 \times 200$, and PGD [53] training (see results in Table V).

While we evaluate performance for the widely considered and challenging L_∞ perturbations, PRIMA can also be used for verifying other specifications including individual fairness [32], global safety properties [8], acoustic [33], geometric [34], and spatial [35] based perturbations.

For classification tasks and ReLU networks, we compare PRIMA with a range of state-of-the-art incomplete verifiers [13–15, 19–22, 24, 55] notably also the ReLU-specialized KPOLY [30], OPTC2V [31], and additionally the highly optimized fully GPU-based β-CROWN [15] (in incomplete mode). For classification using Tanh and Sigmoid activations, fewer verifiers are available and thus we compare with the state-of-the-art incomplete verifier DEEPPOLY [19]. Few verification methods consider the regression setting and to the best of our knowledge, we are the first to analyze the full size `DAVE` network. NEURIFY [13] analyses a heavily scaled down version in a binary classification setting, but in complete mode it does not scale to the much larger networks analysed here. In incomplete mode, it uses the same bounds as DEEPZONO [21] and is less precise than GPUPOLY [29] to which we compare. β-CROWN does not support regression tasks and while an extension might be possible, it is non-trivial. It is also unclear if the approach scales to networks of this size.

C. Image Classification with ReLU activation

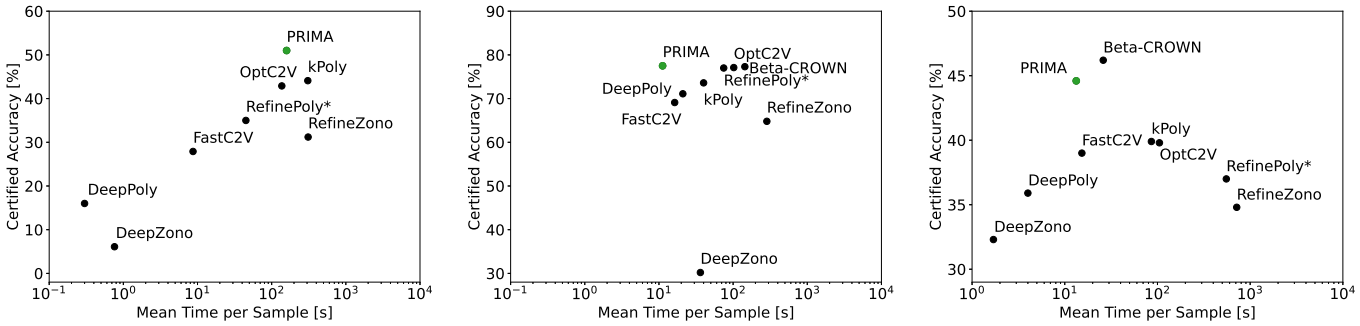
For our experiments, we use a setup similar to KPOLY in [30]. During verification, we use DEEPPOLY (or GPUPOLY [29] for convolutional networks) to determine the octahedral inputs required to compute approximations with our framework. All constraints produced by SBLM are added to the LP

[5]The labels of the original test set are not available (any more), so we used videos 1, 2, 5, and 6 as train and video 4 (instead of 3) as test dataset.

Table II: Number of verified adversarial regions of the first 1000 samples and runtime for PRIMA, OPTC2V, β-CROWN and KPOLY. Natural (NOR), adversarial (PGD [53]), or provable (DiffAI [23], Wong [54]) training was used.

Dataset	Model	Training	Accuracy	ϵ	n_s	KPOLY [30]		OPTC2V [31]		β-CROWN [15]		PRIMA (ours)		# Upper Bound
						# Ver	Time	# Ver	Time	# Ver	Time	# Ver	Time	
MNIST	5×100	NOR	960	0.026	100	441	307	429	137	-	-	**510**	159	842
	8×100	NOR	947	0.026	100	369	171	384	759	-	-	**428**	301	820
	5×200	NOR	972	0.015	50	574	187	601	403	-	-	**690**	224	901
	8×200	NOR	950	0.015	50	506	464	528	3451	-	-	**624**	395	911
	`ConvBig`	DiffAI	929	0.300	100	736	40	771	102	773	144	**775**	11	804
CIFAR10	`ConvSmall`	PGD	630	2/255	100	399	86	398	105	**462**	26	446	13	482
	`ConvBig`	PGD	631	2/255	100	459	346	-	-	**493**	164	483	176	613
	`ResNet`	Wong	290	8/255	50	245	91	-	-	**249**	31	**249**	64	290

- OPTC2V [31]: The code has not been released and no results were reported on these networks. β-CROWN [15] does not yet support MLPs.



(a) MNIST 5×100 (b) MNIST `ConvBig` (c) CIFAR10 `ConvSmall`

Figure 10: Comparison of the runtime/accuracy trade-off of PRIMA (ours), β-CROWN [15], OPTC2V [31], FASTC2V [31], KPOLY [30], REFINEPOLY [19], DEEPPOLY [14] (equivalent bounds to CROWN [22] and CNN-CERT [20]), REFINEZONO [14] and DEEPZONO [21] (equivalent bounds to FAST-LIN [24] and NEURIFY [13] in incomplete mode) on the MNIST 5×100 and `ConvBig` and CIFAR10 `ConvSmall` classifiers, evaluated on the first 1000 samples (100 for REFINEPOLY) of the corresponding test sets. Higher and further left is better.

encoding of the network. After all layers are processed, an LP solver is used to prove the property.

For fully-connected networks, we further refine neuron bounds on each layer as described in [14]. The key idea is to tighten the neuron-wise lower and upper bounds by formulating and solving an LP. For neurons in the second ReLU layer, the MILP encoding from Tjeng et al. [2] is used to tighten the bounds further. We note that encoding more layers with MILP does not scale on these networks. For convolutional networks, we encode some of the neurons in the last one or two fully connected layers using the MILP encoding from [2] and solve the resulting MILP. We note that the concurrent bound optimization in β-CROWN corresponds to simultaneous bound-refinement on all neurons of all layers, which is orthogonal to our approach and is a promising direction to be explored in future work (though intractable without a GPU-based solver).

SBLM is significantly more scalable to bigger groups of neurons than the naive convex hull approach. Already for $k = 4$ ReLU neurons, the standard convex hull computation takes several minutes for a single group, whereas SBLM computes approximations in less than 50 milliseconds. Nevertheless, we find empirically, that the best strategy to leverage this speed-up is to evaluate a large number of small groups. In all our experiments, we consider groups of size $k = 3$ and tune sparse groupings n_s empirically, as shown in Table II.

Comparison with the state-of-the-art. Figure 10 shows a scatter plot comparing the runtime and performance of PRIMA vs. other state-of-the-art verifiers on the robustness certification of the following three classifiers: a normally trained 5×100, a provably trained `ConvBig` (MNIST) and an adversarially trained `ConvSmall` (CIFAR10). β-CROWN is omitted on the first, as we were not able to run it on MLPs. We note that adversarially and provably trained networks sacrifice accuracy for ease of certification, making normally trained networks more relevant and challenging. Here, fast purely propagation based incomplete verifiers like DEEPPOLY verify only about 16% of the images. In contrast, PRIMA verifies 51% in $<$ 160 seconds per image. The closest verifiers in terms of precision are KPOLY and OPTC2V, which verify 44% and 43% of samples and take around 310 and 140 seconds, respectively. Based on these observations, we compare PRIMA

Table III: Number of verified adversarial regions of the 100 random samples from the CIFAR10 test set evaluated by [15]. `CNN-A-Mix` is trained using a combination of adversarial and natural training and `CNN-B-Adv` only adversarially (PGD [53]). Both are taken from [55].

Model	ϵ	Acc	β-CROWN [15]		PRIMA (ours)		# Bound
			# Ver	Time	# Ver	Time	
`CNN-A-Mix`	2/255	100	43	220	**57**	88	69
`CNN-B-Adv`	2/255	100	**46**	234	43	259	83

with KPOLY and OPTC2V on the remaining benchmarks from Singh et al. [30]. Where the concurrently developed β-CROWN can be applied, it is competitive and compared to separately. The recent SDP-based SDP-FO [55] takes many hours per sample and is outperformed by β-CROWN. Thus we do not compare to it.

Comparison with KPOLY and OPTC2V. For all normally trained networks, PRIMA is significantly more accurate than both KPOLY [30] and OPTC2V [31], verifying between 44 and 96 more regions than the better of the two, and sometimes also significantly faster. These results are summarized in Table II. For the DiffAI trained `ConvBig` MNIST network, we verify 4 more regions than OPTC2V. We note again that these networks are comparatively easy to verify (as can be seen in Figure 10b) and therefore we gain less precision. However, the easier proofs come at the cost of reduced accuracy, making them less relevant for real-world applications. For example in Table II, the accuracy of the DiffAI-trained `ConvBig` is the lowest among all MNIST networks. For both PGD-trained CIFAR10 networks, PRIMA verifies between 34 and 63 more regions than KPOLY and OPTC2V while being around four times faster. On the provably trained `ResNet`, PRIMA is faster and marginally more precise than KPOLY however, this network is so heavily regularized that even complete verification is tractable. In summary, PRIMA is usually faster than KPOLY and OPTC2V, especially on larger networks, and is always more precise, sometimes substantially so.

Comparison with β-CROWN. β-CROWN [15] is a highly optimized, fully GPU-based complete BaB [58] solver, supporting only ReLU activations[6] and the classification setting. When comparing complete and incomplete verifiers on accuracy, it is crucial to ensure that similar runtimes were achieved, as complete verifiers can, given sufficient time, decide any property. The GPU-based LP solver underlying β-CROWN is an orthogonal development to the PRIMA multi-neuron constraints. PRIMA currently uses a much slower CPU-based solver which is the main bottleneck as the runtime for computing multi-neuron constraints becomes small via our improved algorithms. We consider combining GPU-based solver from β-CROWN with our multi-neuron approximations as an interesting item for future work. Despite this discrepancy in LP-solver performance distorting the comparison, PRIMA is still significantly faster on some benchmarks at similar (e.g. `ConvBig`) or even notably higher precision (e.g. `CNN-A-Mix`). In other settings (e.g. `CNN-B-Adv` or `ConvBig`), β-CROWN achieves comparable or higher precision in similar runtime.

[6]Extensions to piecewise-linear activations with more than $m = 2$ linear regions would significantly increase runtime ($\mathcal{O}(m^d)$ with split depth d), while precision would be significantly lower for non-piecewise linear activations.

Table IV: Number of verified adversarial regions and runtime in seconds of PRIMA vs. DEEPPOLY for Tanh/Sigmoid on 100 images from the MNIST dataset.

Act.	Model	Acc.	ϵ	DEEPPOLY		PRIMA	
				Ver.	Time	Ver.	Time
Tanh	6×100	97	0.006	38	0.3	**61**	72.5
	9×100	98	0.006	18	0.4	**52**	183.0
	6×200	98	0.002	39	0.6	**68**	170.0
	`ConvSmall`	99	0.005	16	0.4	**30**	27.8
Sigm	6×100	99	0.015	30	0.3	**53**	96.9
	9×100	99	0.015	38	0.5	**56**	336.4
	6×200	99	0.012	43	1.0	**73**	267.0
	`ConvSmall`	99	0.014	30	0.5	**51**	50.0

D. Image Classification with Tanh and Sigmoid activations

While using the exact convex hull algorithm for ReLU relaxations is merely slow, it becomes infeasible for non-piecewise-linear activations such as Tanh and Sigmoid. Computing the constraints for a single group of $k = 3$ neurons can take minutes using the exact convex hull, whereas SBLM using PDDM takes only 10 milliseconds. This dramatic speedup is a result of the SBLM's decompositional approach, solving the problem in lower dimensions (see Section VI), significantly reducing its complexity. Note that both methods compute only approximations for these cases, as the underlying interval-wise bounds are not exact.

We evaluate our method on the MNIST dataset for normally trained, fully-connected and convolutional networks. We choose an ϵ for the B_ϵ^∞ region such that the current state-of-the-art verifier for Tanh and Sigmoid activations, DEEPPOLY, verifies less than 50% of adversarial regions. We remark that DEEPPOLY is based on the same principles and has similar precision as other state-of-the-art verifiers for these activations such as CNN-CERT [20] and CROWN [22].

We use sparse groups of size $n_s = 10$, compute relaxations jointly for $k = 3$ neurons, and again refine neuron-wise lower and upper bounds for fully-connected networks. We verify between 14% and 34% more regions than the current state-of-the-art, in some cases doubling the number of verified samples, while maintaining a reasonable runtime comparable to that for ReLU networks (see Table IV).

E. Autonomous Driving

We evaluate PRIMA in the setting of autonomous driving, or more concretely steering angle prediction, demonstrating scalability to large networks ($>$ 100k neurons and over 27

Table V: Mean absolute steering angle error (smaller is better) for PRIMA vs. GPUPOLY evaluated on every 20^{th} sample and mean evaluation time in seconds.

ϵ	Method	MAE	emp. MAE	cert. MAE	cert. Width	Time
$1/255$	GPUPOLY	7.37°	9.41°	10.35°	5.75°	1.55
	PRIMA	7.37°	9.41°	**10.17°**	**5.29°**	130.1
$2/255$	GPUPOLY	7.37°	11.46°	18.35°	19.63°	2.41
	PRIMA	7.37°	11.46°	**16.82°**	**16.54°**	389.9

Figure 11: Two representative samples from the self-driving car dataset. The target steering angle is illustrated in green, the predicted one in blue. The empirical bounds for $\epsilon = 2/255$ are shown in red and the certified range is shaded blue.

million connections) and inputs ($3 \times 66 \times 200$) of real-world relevance. We certify upper and lower bounds to the predicted steering angle under an L_∞ threat-model, reporting the certified maximum absolute steering angle error as well as the width of reachable steering angles. We use PGD [53] to compute empirical bounds (emp). The CNN architecture proposed by Bojarski et al. [56], adversarial training and the Udacity autonomous driving dataset [57] are used.

When the permissible perturbation size is small and the standard error of the model is larger than the perturbation effect, cheaper methods such as GPUPOLY already yield good results. However, for slightly larger perturbations, PRIMA reduces the gap between empirical and certified error by over 20% (see Table V). In Figure 11, we show two representative samples from the dataset, where the certified region for $\epsilon = 2/255$ is shaded blue, the empirical bounds are shown in red, the target in green and the prediction in blue.

F. Effectiveness of SBLM approximation

Computing approximations with SBLM using PDDM has two main advantages compared to the direct convex hull approach: It is significantly faster and produces fewer constraints, making the resulting LP easier to solve, while barely losing any precision.

For example, verifying the 5×100 network with PRIMA and comparing the 3-ReLU relaxations computed with the exact convex hull and SBLM to the neuron-wise 1-ReLU approximations (Figure 12), we observe: SBLM is on average 200 times faster than the exact convex hull, while being only marginally less precise. That is, the volume of the constraint polytopes in the 6-dimensional input-output space of individual neuron groups is only a few percent larger using SBLM. In contrast, both multi-neuron approaches yield on average three times smaller volumes than 1-ReLU approximations.

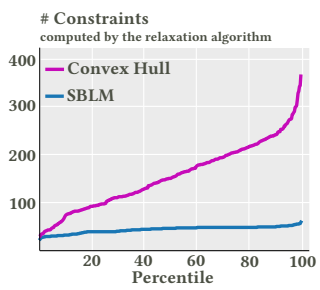
In addition to the faster generation of constraints, the runtime analysis in Figure 13 shows that using SBLM instead of the exact convex hull also speeds up LP solving as 4-times fewer constraints are generated, leading to a total 8-fold runtime reduction, while verifying the same number of images. This effect is also observed in the additionally performed, time-intensive neuron refinement, where SBLM reduces the runtime by 70% while verifying 3% more images. Using a larger number of the more diverse SBLM constraints leads to tighter neuron-wise bounds, allowing an earlier termination of the LP solver and explaining the accuracy improvement. Using SBLM with neuron refinement is in fact still quicker than the standard, exact convex hull computation.

The combination of a much faster computation and a four-fold reduction in returned constraints allows us to consider many more neuron groups and consequently discover more diverse constraints when using SBLM. This enables the LP solver to compute tighter bounds faster, despite only using approximate constraints.

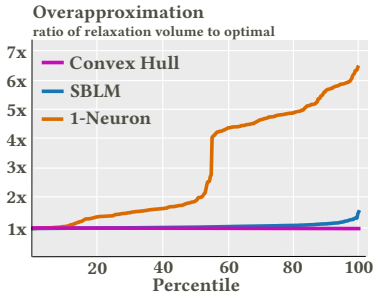
VIII. RELATED WORK

The importance of certifying the robustness of neural networks to input perturbations has created a surge of research activity in recent years. The approaches with deterministic guarantees can be divided into exact and incomplete methods. Incomplete methods are much faster and more scalable than exact ones, but they can be imprecise, i.e., they may fail to certify a property even if it holds.

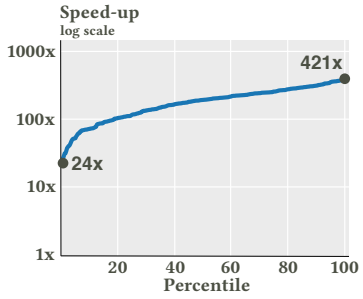
Complete methods are mostly based on satisfiability modulo theory (SMT) [6–9] or the branch-and-bound approach [2–5, 10, 12, 15, 16], often implemented using mixed integer linear programming (MILP). These methods offer exactness guarantees but are based on solving NP-hard optimization problems, which can make them intractable even for small networks. Incomplete methods can be divided into bound propagation approaches [19, 21–24, 29, 59] and those that generate polynomially-solvable optimization problems [17, 30, 31, 55, 60–62] such as linear programming (LP) or semidefinite programming (SDP) optimization problems. Compared to deterministic certification methods, randomized smoothing [63–65] can only provide probabilistic guarantees, incurs significant runtime costs at inference, and generalization to arbitrary safety properties is still an open problem.



(a) Number of constraints for individual k-neuron abstractions.



(b) Volume of the constraint polytope in comparison to the exact convex hull.



(c) Speedup realized by using SBLM compared to exact convex hull.

Figure 12: Case study: Analysis of the distribution of the number of discovered constraints, abstraction volume, and runtime over all (≈ 400) individual 3-neuron groups processed during verification of a single MNIST image on 5×100 ReLU network.

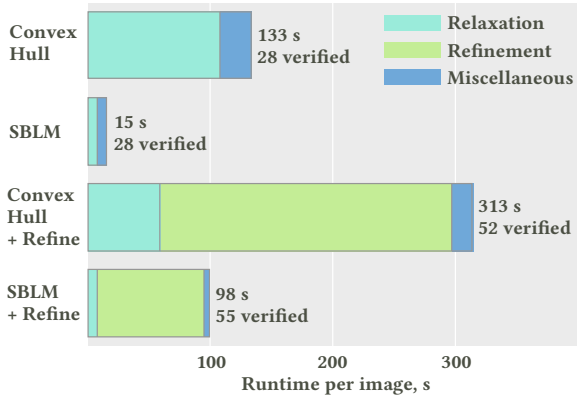


Figure 13: Runtime comparison of using SBLM vs. exact convex hull for computing relaxations in PRIMA. Evaluated on 100 images and the MNIST 5×100 ReLU network.

A new avenue towards more precision are methods [30, 31] breaking the so-called convex barrier [25] by considering activation functions jointly. However, their scalability is limited by the need to solve NP-hard convex hull problems. There are many approaches for solving the convex hull problem for polyhedra exactly [39–41, 66–70], in contrast to the few approximate methods either sacrificing soundness [44–47] or exhibiting exponential complexity [48], prohibiting their use in neural network verification.

Our work follows the line of convex barrier-breaking methods, generalizing the concept to arbitrary bounded, multivariate activations. In contrast to prior work, we decompose the underlying convex hull problem into lower-dimensional spaces and solve it approximately using a novel relaxed Double Description, irredundancy formulation, and a new ray-shooting-based algorithm to add multiple constraints jointly. The resulting speed-ups make PRIMA tractable for non-piecewise-linear activations, a first for convex barrier-breaking methods.

IX. CONCLUSION

We presented PRIMA, a general framework that substantially advances the state-of-the-art in neural network verification by providing efficient multi-neuron abstractions for arbitrary, bounded, multivariate non-linear activation functions. Our key idea is to decompose the bottleneck convex hull computation into lower-dimensional spaces, solve it approximately, and leveraging the resulting speedup to evaluate more neuron groups, thus discovering more diverse constraints. Our extensive evaluation shows significant improvements both in precision and speed over prior work.

References

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *Proc. International Conference on Learning Representations (ICLR)*, 2014.

[2] V. Tjeng, K. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," *arXiv preprint arXiv:1711.07356*, 2017.

[3] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener, "Efficient verification of relu-based neural networks via dependency analysis." in *AAAI*, 2020, pp. 3291–3299.

[4] R. Bunel, J. Lu, I. Turkaslan, P. Kohli, P. Torr, and P. Mudigonda, "Branch and bound for piecewise linear neural network verification," *Journal of Machine Learning Research*, vol. 21, no. 2020, 2020.

[5] J. Lu and M. P. Kumar, "Neural network branching for neural network verification," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=B1evfa4tPB

[6] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić *et al.*, "The marabou framework for verification and analysis of deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2019, pp. 443–452.

[7] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2017, pp. 269–286.

[8] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 97–117.

[9] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 3–29.

[10] R. Anderson, J. Huchette, W. Ma, C. Tjandraatmadja, and J. P. Vielma, "Strong mixed-integer programming formulations for trained neural networks," *Mathematical Programming*, pp. 1–37, 2020.

[11] G. Anderson, S. Pailoor, I. Dillig, and S. Chaudhuri, "Optimization and abstraction: A synergistic approach for analyzing neural network robustness," in *Proc. Programming Language Design and Implementation (PLDI)*, 2019, p. 731–744.

[12] A. D. Palma, H. S. Behl, R. Bunel, P. H. S. Torr, and M. P. Kumar, "Scaling the convex barrier with sparse dual algorithms," 2021.

[13] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Efficient formal safety analysis of neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 6367–6377.

[14] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "Boosting robustness certification of neural networks," in *International Conference on Learning Representations*, 2019.

[15] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter, "Beta-crown: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification," *arXiv preprint arXiv:2103.06624*, 2021.

[16] K. Xu, H. Zhang, S. Wang, Y. Wang, S. Jana, X. Lin, and C.-J. Hsieh, "Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers," *arXiv preprint arXiv:2011.13824*, 2020.

[17] Z. Lyu, C.-Y. Ko, Z. Kong, N. Wong, D. Lin, and L. Daniel, "Fastened crown: Tightened neural network robustness certificates," *arXiv preprint arXiv:1912.00574*, 2019.

[18] E. Wong and Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5286–5295.

[19] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–30, 2019.

[20] A. Boopathy, T.-W. Weng, P.-Y. Chen, S. Liu, and L. Daniel, "Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks," in *AAAI Conference on Artificial Intelligence (AAAI)*, Jan 2019.

[21] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and effective robustness certification," *Advances in Neural Information Processing Systems*, vol. 31, pp. 10 802–10 813, 2018.

[22] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," in *Advances in neural information processing systems*, 2018.

[23] M. Mirman, T. Gehr, and M. Vechev, "Differentiable abstract interpretation for provably robust neural networks," in *International Conference on Machine Learning*, 2018.

[24] T.-W. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, D. Boning, I. S. Dhillon, and L. Daniel, "Towards fast computation of certified robustness for relu networks," *arXiv preprint arXiv:1804.09699*, 2018.

[25] H. Salman, G. Yang, H. Zhang, C.-J. Hsieh, and P. Zhang, "A convex relaxation barrier to tight robustness verification of neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 9835–9846.

[26] H. Tran, S. Bak, W. Xiang, and T. T. Johnson, "Verification of deep convolutional neural networks using imagestars," in *Proc. Computer Aided Verification (CAV)*, S. K. Lahiri and C. Wang, Eds., 2020, pp. 18–42.

[27] K. Xu, Z. Shi, H. Zhang, Y. Wang, K. Chang, M. Huang, B. Kailkhura, X. Lin, and C. Hsieh, "Automatic perturbation analysis for scalable certified robustness and

beyond," 2020.

[28] C. Müller, F. Serre, G. Singh, M. Püschel, and M. Vechev, "Scaling polyhedral neural network verification on gpus," *Proc. Machine Learning and Systems (MLSys)*, 2021.

[29] C. Müller, G. Singh, M. Püschel, and M. Vechev, "Neural network robustness verification on gpus," 2020.

[30] G. Singh, R. Ganvir, M. Püschel, and M. Vechev, "Beyond the single neuron convex barrier for neural network certification," in *Advances in Neural Information Processing Systems*, 2019, pp. 15 098–15 109.

[31] C. Tjandraatmadja, R. Anderson, J. Huchette, W. Ma, K. Patel, and J. P. Vielma, "The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification," *arXiv preprint arXiv:2006.14076*, 2020.

[32] A. Ruoss, M. Balunović, M. Fischer, and M. Vechev, "Learning certified individually fair representations," *arXiv preprint arXiv:2002.10312*, 2020.

[33] W. Ryou, J. Chen, M. Balunovic, G. Singh, A. Dan, and M. Vechev, "Fast and effective robustness certification for recurrent neural networks," *arXiv preprint arXiv:2005.13300*, 2020.

[34] M. Balunović, M. Baader, G. Singh, T. Gehr, and M. Vechev, "Certifying geometric robustness of neural networks," *Advances in Neural Information Processing Systems 32*, 2019.

[35] A. Ruoss, M. Baader, M. Balunović, and M. Vechev, "Efficient certification of spatial robustness," *arXiv preprint arXiv:2009.09318*, 2020.

[36] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "Ai2: Safety and robustness certification of neural networks with abstract interpretation," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 3–18.

[37] B. Chazelle, "An optimal convex hull algorithm in any fixed dimension," *Discrete & Computational Geometry*, vol. 10, no. 4, pp. 377–409, 1993.

[38] R. Seidel, "The upper bound theorem for polytopes: an easy proof of its asymptotic version," *Computational Geometry*, vol. 5, no. 2, pp. 115–116, 1995.

[39] H. Edelsbrunner, *Algorithms in combinatorial geometry*. Springer Science & Business Media, 2012, vol. 10.

[40] T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall, "The double description method," *Contributions to the Theory of Games*, vol. 2, no. 28, pp. 51–73, 1953.

[41] K. Fukuda and A. Prodon, "Double description method revisited," in *Franco-Japanese and Franco-Chinese Conference on Combinatorics and Computer Science*. Springer, 1995, pp. 91–111.

[42] G. Singh, M. Püschel, and M. Vechev, "Fast polyhedra abstract domain," in *Proc. Principles of Programming Languages (POPL)*, 2017, pp. 46–59.

[43] R. Clarisó and J. Cortadella, "The octahedron abstract domain," *Science of Computer Programming*, vol. 64, no. 1, pp. 115–139, 2007.

[44] J. L. Bentley, F. P. Preparata, and M. G. Faust, "Approximation algorithms for convex hulls," *Communications of the ACM*, vol. 25, no. 1, pp. 64–68, 1982.

[45] H. R. Khosravani, A. E. Ruano, and P. M. Ferreira, "A simple algorithm for convex hull determination in high dimensions," in *2013 IEEE 8th International Symposium on Intelligent Signal Processing*. IEEE, 2013, pp. 109–114.

[46] J. Zhong, K. Tang, and A. K. Qin, "Finding convex hull vertices in metric space," in *2014 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2014, pp. 1587–1592.

[47] H. Sartipizadeh and T. L. Vincent, "Computing the approximate convex hull in high dimensions," *arXiv preprint arXiv:1603.04422*, 2016.

[48] Z.-B. Xu, J.-S. Zhang, and Y.-W. Leung, "An approximate algorithm for computing multidimensional convex hulls," *Applied mathematics and computation*, vol. 94, no. 2-3, pp. 193–226, 1998.

[49] B. Genov, "The convex hull problem in practice: improving the running time of the double description method," Ph.D. dissertation, 2015.

[50] K. Fukuda, "Polyhedral computation," Zurich, 2020-07-10.

[51] A. Maréchal and M. Périn, "Efficient elimination of redundancies in polyhedra using raytracing," 2017.

[52] Gurobi Optimization, LLC, "Gurobi optimizer reference manual," 2018. [Online]. Available: http://www.gurobi.com

[53] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[54] E. Wong, F. R. Schmidt, J. H. Metzen, and J. Z. Kolter, "Scaling provable adversarial defenses," *arXiv preprint arXiv:1805.12514*, 2018.

[55] S. Dathathri, K. Dvijotham, A. Kurakin, A. Raghunathan, J. Uesato, R. R. Bunel, S. Shankar, J. Steinhardt, I. Goodfellow, P. S. Liang *et al.*, "Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[56] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[57] "Using deep learning to predict steering angles," https://github.com/udacity/self-driving-car.

[58] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell, "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning," *Discrete Optimization*, vol. 19, pp. 79–102, 2016.

[59] S. Gowal, K. D. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, R. Arandjelovic, T. Mann, and P. Kohli, "Scalable verified training for provably robust image classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 4842–4851.

[60] R. R. Bunel, O. Hinder, S. Bhojanapalli, and K. Dvijotham, "An efficient nonconvex reformulation of stagewise convex optimization problems," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[61] A. Raghunathan, J. Steinhardt, and P. S. Liang, "Semidefinite relaxations for certifying robustness to adversarial examples," in *Advances in Neural Information Processing Systems*, 2018, pp. 10 877–10 887.

[62] W. Xiang, H.-D. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multilayer neural networks," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 11, pp. 5777–5783, 2018.

[63] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," *2019 IEEE Symposium on Security and Privacy (S&P)*, 2018.

[64] J. Cohen, E. Rosenfeld, and Z. Kolter, "Certified adversarial robustness via randomized smoothing," in *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[65] H. Salman, G. Yang, J. Li, P. Zhang, H. Zhang, I. Razenshteyn, and S. Bubeck, "Provably robust deep learning via adversarially trained smoothed classifiers," *arXiv preprint arXiv:1906.04584*, 2019.

[66] M. Joswig, "Beneath-and-beyond revisited," in *Algebra, Geometry and Software Systems*. Springer, 2003, pp. 1–21.

[67] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hull," Technical Report GCG53, The Geometry Center, MN, Tech. Rep., 1993.

[68] G. B. Dantzig, *Linear programming and extensions*. Princeton university press, 1998, vol. 48.

[69] D. Avis and K. Fukuda, "A basis enumeration algorithm for linear systems with geometric applications," *Applied Mathematics Letters*, vol. 4, no. 5, pp. 39–42, 1991.

[70] ——, "A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra," *Discrete & Computational Geometry*, vol. 8, no. 3, pp. 295–313, 1992.

APPENDIX A
PARTIAL DOUBLE DESCRIPTION METHOD PROOFS

To proof the exactness guarantee given for the intersection of two polytopes using our batch intersection and precision boosting approach, described in Section V-B, we first have to proof the following results on batch intersection (Algorithm 1) and precision boosting (Algorithm 2):

Algorithm 1: Batch Intersection

Result: Intersected cone $(\boldsymbol{A} \cup \boldsymbol{A}', \mathcal{R}')$
Input: Cone $(\boldsymbol{A}, \mathcal{R})$, constraint matrix $\boldsymbol{A}'$
Initialize $\mathcal{R}_-, \mathcal{R}_0, \mathcal{R}_+, \mathcal{R}_* = \emptyset, \emptyset, \emptyset, \emptyset$
for r *in* $\mathcal{R}$ **do**
  if $min(\boldsymbol{A}'\boldsymbol{r}) < 0$ **then**
    Add r to $\mathcal{R}_-$
  else if $min(\boldsymbol{A}'\boldsymbol{r}) > 0$ **then**
    Add r to $\mathcal{R}_+$
  else
    Add r to $\mathcal{R}_0$
end
for r_+ *in* $\mathcal{R}_+$ **do**
  for r_- *in* $\mathcal{R}_-$ **do**
    Compute r_* using ray-shooting from r_+ to r_-
    Add r_* to $\mathcal{R}_*$
  end
end
Construct new PDD $(\boldsymbol{A} \cup \boldsymbol{A}', \mathcal{R}_0 \cup \mathcal{R}_+ \cup \mathcal{R}_*)$
Make PDD A-irredundant
return *PDD*

Batch intersection following Algorithm 1 of a cone in DD with a matrix of constraints yields the following guarantee for the resulting PDD:

Theorem A.1. *Given a Double Description $(\boldsymbol{A}, \mathcal{R})$ of a polyhedral cone and the constraint matrix $\boldsymbol{A}'$, adding all constraints jointly as per Algorithm 1 is guaranteed to yield a double description $(\boldsymbol{A} \cup \boldsymbol{A}', \mathcal{R}')$ enumerating* all *extremal rays r' of the $\boldsymbol{A} \cup \boldsymbol{A}'$-induced cone with one of the following properties:*

1) *r' is extremal (rank $d-1$) in the $\boldsymbol{A}$-induced cone.*
2) *r' is of rank $d-2$ in the $\boldsymbol{A}$-induced cone.*

Proof. We can formally divide the rays of the new PDD $\mathcal{R}'$ into the two non-overlapping sets:

- $\mathcal{R}_+ \cup \mathcal{R}_0$ – Rays in $\mathcal{R}$ not violating any constraint $a \in \boldsymbol{A}'$
- $\mathcal{R}_*$ – Rays discovered by ray-shooting

Since $(\boldsymbol{A}, \mathcal{R})$ is a DD of the $\boldsymbol{A}$-induced cone it enumerates all extremal rays. If r' is extremal in both the $\boldsymbol{A}$-induced and the $\boldsymbol{A} \cup \boldsymbol{A}'$-induced cones it is included in $\mathcal{R}$ and does not violate any constraints. Therefore, it is included in the first group above and will be part of $\mathcal{R}'$, which concludes the proof of the first point. Any ray of rank $d-2$ can by definition be represented as a positive combination of two extremal rays, that is rays of rank $d-1$. As we assume ray r' to be extremal in the $\boldsymbol{A} \cup \boldsymbol{A}'$-induced cone and therefore have rank $d-1$ it necessarily intersects at least one constraint $\boldsymbol{a} \in \boldsymbol{A}'$ and is extremal to the $\boldsymbol{A} \cup \boldsymbol{a}$-induced cone. Consequently exactly one of the extremal rays used to construct it has to lie on either side of thy hyperplane induced by constraint $\boldsymbol{a}$. Therefore, they will be included in the sets $\mathcal{R}_+$ and $\mathcal{R}_-$ and the intersection will be discovered as part of the ray-shooting, concluding the proof of the second point. □

Algorithm 2: PDDM Intersection

Result: Intersected cone $(\boldsymbol{A}_1 \cup \boldsymbol{A}_2, \mathcal{R}')$
Input: Cones $(\boldsymbol{A}_1, \mathcal{R}_1)$ and $(\boldsymbol{A}_2, \mathcal{R}_2)$
Compute $(\boldsymbol{A}', \mathcal{R}_1') = (\boldsymbol{A}_1 \cup \boldsymbol{A}_2, \mathcal{R}_1)$ with Algorithm 1
Compute $(\boldsymbol{A}', \mathcal{R}_2') = (\boldsymbol{A}_2 \cup \boldsymbol{A}_1, \mathcal{R}_2)$ with Algorithm 1
Construct new PDD $(\boldsymbol{A}', \mathcal{R}_1' \cup \mathcal{R}_2')$
Make PDD A-irredundant
return *PDD*

Using this result, we can proof the following guarantee for intersections of two cones in DD using our batch intersection and precision boosting approach, described in Section V-B and Algorithm 2:

Theorem A.2. *Given the double descriptions $(\boldsymbol{A}_1, \mathcal{R}_1)$ and $(\boldsymbol{A}_2, \mathcal{R}_2)$ of two polyhedral cones, their intersection computed as per Algorithm 2 is guaranteed to be a partial double description $(\boldsymbol{A}_1 \cup \boldsymbol{A}_2, \mathcal{R}')$ enumerating* all *extremal rays r' of the $(\boldsymbol{A}_1 \cup \boldsymbol{A}_2)$-induced cone with one of the following properties:*

1) *r' is extremal in the $\boldsymbol{A}_1$-induced cone.*
2) *r' is extremal in the $\boldsymbol{A}_2$-induced cone.*
3) *r' is of rank $d-2$ in the $\boldsymbol{A}_1$-induced cone.*
4) *r' is of rank $d-2$ in the $\boldsymbol{A}_2$-induced cone.*

Proof. The proof follows directly from applying Lemma A.1 to both applications of Algorithm 1, the insight that every extremal ray discovered by either will be included in the final generating set $\mathcal{R}'$ and the observation that the intersection of the exact $\mathcal{H}$-representations, trivially is the union of their respective constraints, leading to a valid partial double description. □

Using these results, we can in turn proof the following exactness guarantee for the intersection of two polyhedral cones of up to dimension 4 in DD using the approach described in Section V-B:

Theorem A.3. *Given the Double Description $(\boldsymbol{A}_1, \mathcal{R}_1)$ and $(\boldsymbol{A}_2, \mathcal{R}_2)$ of two polyhedral cones $\mathcal{P}_\infty$ and $\mathcal{P}_\in$ of dimension $d \leq 4$, the PDD of their intersection $(\boldsymbol{A}_1 \cup \boldsymbol{A}_2, \mathcal{R}')$ as computed with Algorithm 2 is in fact a DD with an irredundant generating set $\mathcal{R}'$.*

Proof. For briefness sake, we will only show the proof for $d = 4$ here. Let $\mathcal{R}^*$ be the set of extremal rays of the $(\boldsymbol{A}_1 \cup \boldsymbol{A}_2)$-induced polyhedral cone. Consequently $\boldsymbol{r}^* \in \mathcal{R}^*$ has the rank $d - 1 = 3$ in this cone and therefore it fulfills 3 linearly

independent constraints in $\boldsymbol{A}_1 \cup \boldsymbol{A}_2$ with equality. This leads to the following four exhaustive options:

1) All 3 constraints are part of $\boldsymbol{A}_1$, $\boldsymbol{r}^*$ is extremal in $\mathcal{P}_\infty$
2) All 3 constraints are part of $\boldsymbol{A}_2$, $\boldsymbol{r}^*$ is extremal in $\mathcal{P}_\in$
3) 2 constraints are part of $\boldsymbol{A}_1$ and 1 of $\boldsymbol{A}_2$, $\boldsymbol{r}^*$ is of rank $d-2=2$ in P_1
4) 2 constraints are part of $\boldsymbol{A}_2$ and 1 of $\boldsymbol{A}_1$, $\boldsymbol{r}^*$ is of rank $d-2=2$ in P_2

As all of those are enumerated by Algorithm A.2, $\mathcal{R}'$ will therefore include all extremal rays of the $(\boldsymbol{A}_1 \cup \boldsymbol{A}_2)$-induced cone. In this case A-irredundancy is equivalent with irredundancy, concluding the proof. □

APPENDIX B
PDDM COMPLEXITY

In this section, we analyse the complexity of our approximate Partial Double Description Method for the computation of an over-approximation to the convex hull:

Theorem B.1. *Given the DD of two d-dimensional, bounded polytopes generated by at most n_v vertices or equivalently induced by at most n_a constraints, computing a sound over-approximation of their convex hull as per Algorithm 2 has a worst-case time complexity of:*

$$\mathcal{O}(n_v \cdot n_a^4 + n_a^2 \log(n_a^2)). \tag{7}$$

Proof. The PDDM can be broken down in its six components illustrated in Figure 5:

1) Conversion from primal to dual representation (Section V-A)
2) Adding the constraints of one polytope to the other, or more concretely separation of vertices into the three sets $\mathcal{R}_+$, $\mathcal{R}_0$, and $\mathcal{R}_-$ (Section V-B or first half of Algorithm 1)
3) Discovery of new vertices via ray-shooting (Section V-B or second half of Algorithm 1)
4) Combining the vertices of the two intersection directions (Section V-B or Algorithm 2)
5) Enforcing of A-irredundancy (Section V-C or Algorithm 2)
6) Conversion from dual to primal representation (Section V-A)

Primal-dual conversions and combining of vertices are computed in constant time, as this only involves computing the transpose and concatenation which can be done implicitly by changing the indexing of the corresponding matrices. Therefore, we will focus on the remaining three steps, which are all conducted in dual space.

In the following we assume the setting, of two d-dimensional, bounded polytopes which in dual-space are defined by $\mathcal{P}_1 = (\boldsymbol{A}_1, \mathcal{R}_1)$ and $\mathcal{P}_2 = (\boldsymbol{A}_2, \mathcal{R}_2)$. For convenience sake, we assume the number of vertices to be $n_v = \max(|\mathcal{R}_1|, |\mathcal{R}_2|)$ and number of constraints $n_a = \max(|\boldsymbol{A}_1|, |\boldsymbol{A}_2|)$. Note that their roles are reversed compared to a primal space representation.

Adding Constraints and Separating Vertices. Recall that in dual space we compute the intersection of the two polytopes $\mathcal{P}_1$ and $\mathcal{P}_2$. The first step to intersect $\mathcal{P}_1$ with $\mathcal{P}_2$ is to split all points in $\mathcal{R}_1$ into the three groups $\mathcal{R}_+$, $\mathcal{R}_0$, and $\mathcal{R}_-$ defined in Section V-B depending on whether the lie inside, on the border or outside of the polytope defined by $\boldsymbol{A}_2$ as per the first half of Algorithm 1. This requires (at worst) evaluating $\boldsymbol{a}_i \boldsymbol{r}_j - b_i \{>, =, <\} 0$ for all $\boldsymbol{r}_j \in \mathcal{R}_1$ and $\boldsymbol{a}_i, b_i \in \boldsymbol{A}_2$. The addition and comparison involved are dominated by the d-dimensional dot-product between $\boldsymbol{a}_i$ and $\boldsymbol{r}_j$, leading to a total complexity of this step of order $\mathcal{O}(d \cdot n_a \cdot n_v)$. Note that incidence matrix columns corresponding to the new constraints are added and populated without any extra computation with 0s for the vertices in $\mathcal{R}_+$ and 1s for vertices in $\mathcal{R}_0$.

Ray-Shooting. Recall that to discover new generating vertices, the first intersections between the rays shot from all generating vertices of $\mathcal{P}_1$ lying inside $\mathcal{P}_2$, $\boldsymbol{r}_+ \in \mathcal{R}_+$, to all points lying outside $\mathcal{P}_2$, $\boldsymbol{r}_- \in \mathcal{R}_-$, and all constraints in $\boldsymbol{A}_2$ are computed. At worst there are no points in group $\mathcal{R}_0$ and all vertices are spread equally between $\mathcal{R}_+$ and $\mathcal{R}_-$, leading to $\frac{n_v^2}{4}$ rays to be intersected with n_a constraints where each intersection corresponds to computing a ratio of dot-products and is order $\mathcal{O}(d)$. Selecting the first intersection for each ray is linear in the intersection number. Consequently, the ray-shooting process overall is $\mathcal{O}(d \cdot n_a \cdot n_v^2)$. Note that this adds new incidence matrix rows corresponding to the new vertices $\mathcal{R}_*$, which can then be populated with the row obtained by the elementwise and of the two points generating the ray and a 1 in the column associated with the constraint of the first intersection which is linear $\mathcal{O}(n_v)$ and dominated by the previous term.

Enforcing A-Irredundancy. The intermediate state prior to enforcing A-irredundancy contains at most $n = 2(n_v + \frac{n_v^2}{4})$ vertices, consisting of the at most n_v vertices in $\mathcal{R}_+$ and the at most $\frac{n_v^2}{4}$ vertices in $\mathcal{R}_*$, discovered during ray shooting, for both intersection directions. To enforce A-irredundancy, vertices are first sorted in descending order by the number of active constraints which is order $\mathcal{O}(n \log(n))$. Then starting with the first vertex, row-wise inclusion of the corresponding incidence matrix rows is checked for all following elements. Each check is $\mathcal{O}(n_a)$ and $\frac{n^2-n}{2}$ checks have to be performed in the worst case that is, if no element is removed. This leads to an overall complexity of $\mathcal{O}(n_a \cdot n_v^4 + n_v^2 \log(n_v^2))$ for enforcing A-irredundancy.

PDDM Complexity. Putting the three elements together and observing $d < n_v$ for any d-dimensional, bounded polytope, we observe that both the ray-shooting and the separation of vertices get dominated by the last step of enforcing A-irredundancy. Swapping the roles of n_v and n_a to derive an expression in terms of primal space entities, we arrive at an overall complexity of $\mathcal{O}(n_v \cdot n_a^4 + n_a^2 \log(n_a^2))$. □

APPENDIX C
PDDM SOUNDNESS

In this section, we proof the soundness of the Partial Double Description Method. Computing sound over-approximations

of the convex hull of two polytopes in primal space, by inclusion-inversion, is equivalent to computing sound under-approximation of the intersection of their dual space representations.

Since the primal-dual conversion employed in the PDDM is exact, showing a sound under-approximating intersection of two polytopes in PDD in dual space is sufficient for overall soundness.

Enforcing A-irredundancy on a polytope $\mathcal{P}$ to yield $\mathcal{Q}$ can only remove generators, yielding $\mathcal{Q} \subseteq \mathcal{P}$. It follows directly that $\mathcal{Q}$ is a sound under-approximation, if $\mathcal{P}$ is.

If both polytopes $\mathcal{P}_1$ and $\mathcal{P}_2$ generated by the vertex sets obtained for the two directions of batch intersection are sound under-approximations of the true intersection of the exact $\mathcal{H}$-representations, it follows that their union $\mathcal{P}$ is also a sound under-approximation. Hence, the soundness of the PDDM follows if we can show soundness of the batch intersection step.

Theorem C.1. *The intersection $\mathcal{P}' = (\boldsymbol{A}', \mathcal{R}'_p)$ of a polytope $\mathcal{P}$ in PDD $(\boldsymbol{A}_p, \mathcal{R}_p)$ with the exact constraints $\boldsymbol{A}_q$ of a polytope $\mathcal{Q}$ computed with the Batch Intersection, described in Section V-B and detailed in Algorithm 1, is a sound under-approximation of the intersection of the two exact $\mathcal{H}$-representations $\boldsymbol{A}_p$ and $\boldsymbol{A}_q$:*

$$\{\boldsymbol{x} \in \mathbb{R}^d | \boldsymbol{A}'\boldsymbol{x} \geq 0\} = \{\boldsymbol{x} \in \mathbb{R}^d | \boldsymbol{A}_p\boldsymbol{x} \geq 0 \wedge \boldsymbol{A}_q\boldsymbol{x} \geq 0\}$$

$$\left\{ \sum_{\boldsymbol{r}_i \in \mathcal{R}'_p} \lambda_i \boldsymbol{r}_i | \sum_i \lambda_i \leq 1, \lambda_i \in \mathbb{R}_0^+ \right\} \subseteq \{\boldsymbol{x} \in \mathbb{R}^d | \boldsymbol{A}_p\boldsymbol{x} \geq 0 \wedge \boldsymbol{A}_q\boldsymbol{x} \geq 0\}$$

Proof. Recall that a PDD consists of an exact $\mathcal{H}$-representation and an under-approximate $\mathcal{V}$-representation. The intersection of two polytopes in $\mathcal{H}$-representation is simply the union of all constraints, allowing for an exact intersection of the $\mathcal{H}$-representations. Hence, it remains to show that the resulting $\mathcal{V}$-representation ${\mathcal{R}'}_p$ is a sound under-approximation of the $\mathcal{H}$-representation $\boldsymbol{A}'$. For this it is sufficient to show that, by construction, every vertex included in the $\mathcal{V}$-representation $\mathcal{R}'_p$ satisfies all constraints of the $\mathcal{H}$-representation $\boldsymbol{A}'$.

Recall that $\mathcal{R}'_p$ is the union of three groups of vertices (see Section V-B):

- $\mathcal{R}_+$ vertices of the generating set $\mathcal{R}_p$ that satisfy all constraints in $\boldsymbol{A}_q$ strictly,
- $\mathcal{R}_0$ vertices of the generating set $\mathcal{R}_p$ that satisfy all constraints in $\boldsymbol{A}_q$, at least one with equality,
- $\mathcal{R}_*$ the first intersections $\boldsymbol{r}_*$ of rays from a vertex in $\boldsymbol{r}_+ \in \mathcal{R}_+$ to a vertex in $\boldsymbol{r}_- \in \mathcal{R}_-$ (vertices in $\mathcal{R}_p$ not satisfying at least one constraint in $\boldsymbol{A}_q$) with the hyperplanes defined by a constraint in $\boldsymbol{A}_q$. Since $\boldsymbol{r}_-$ lies outside $\mathcal{Q}$ while $\boldsymbol{r}_+$ lies inside, an intersection $\boldsymbol{r}_*$ is guaranteed to exist and lie between the two. Therefore by convexity of $\mathcal{P}$, $\boldsymbol{r}_*$ satisfies all constraints of $\boldsymbol{A}_p$. Further, since $\boldsymbol{r}_*$ is the first intersection of the ray with a constraint in $\boldsymbol{A}_q$ as seen from $\boldsymbol{r}_+$, which satisfies all constraints in $\boldsymbol{A}_q$, $\boldsymbol{r}_*$ also satisfies all constraints in $\boldsymbol{A}_q$.

Consequently, all vertices in the generating set $\mathcal{R}'_p$ satisfy all constraints of both $\mathcal{P}$ and $\mathcal{Q}$. It follows that they generate a polytope that is a subset of the intersection $\mathcal{Q} \cap \mathcal{P}$ and therefore soundly under-approximating it, concluding the soundness proof. □

APPENDIX D
SPLIT-BOUND-LIFT METHOD ALGORITHM

In Algorithm 3 we formalize how the SBML is applied recursively given an ordering of output variables $\mathcal{I}$, a polytope $\mathcal{P}$ constraining the group inputs, a set of bounding regions $\mathcal{D}$ and the associated bounds $\mathcal{B}$.

Algorithm 3: Split-Bound-Lift Method (`SBLM`)

Input: Variable ordering $\mathcal{I}$, input polytope $\mathcal{P}$, set of bounding regions $\mathcal{D}$ and set of bounds $\mathcal{B}$
Output: Jointly constraining polytope $\mathcal{K}$
if $|\mathcal{I}| > 0$ **then**
  Get next output variable: $y \leftarrow \mathcal{I}_0$
  foreach $\mathcal{D}^i, \mathcal{B}^i$ *in* $\mathcal{D}, \mathcal{B}$ **do**
    Create split region: $\mathcal{P}_i = \mathcal{P} \cap \mathcal{D}^i$
    Apply SBLM: $\mathcal{K}_i \leftarrow \texttt{SBLM}(\mathcal{I}_{1:end}, \mathcal{P}_i, \mathcal{D}, \mathcal{B})$
    Extend into space including y: $\mathcal{K}_i \leftarrow \mathcal{K}_i \times \mathbb{R}$
    Apply bounds $\mathcal{B}^i$: $\mathcal{K}_i \leftarrow \mathcal{K}_i \cap \mathcal{B}^i$
  end
  Compute convex hull: $\mathcal{K} = \texttt{PDDM}(\{\mathcal{K}_i\}_i)$
  return $\mathcal{K}$
else
  return $\mathcal{P}$
end