ReLU Hull Approximation

ZHONGKUI MA, The University of Queensland, Australia
JIAYING LI, Microsoft, China
GUANGDONG BAI, The University of Queensland, Australia

Convex hulls are commonly used to tackle the non-linearity of activation functions in the verification of neural networks. Computing the exact convex hull is a costly task though. In this work, we propose a fast and precise approach to over-approximating the convex hull of the ReLU function (referred to as the *ReLU hull*), one of the most used activation functions. Our key insight is to formulate a *convex polytope* that "wraps" the ReLU hull, by reusing the linear pieces of the ReLU function as the lower faces and constructing upper faces that are adjacent to the lower faces. The upper faces can be efficiently constructed based on the edges and vertices of the lower faces, given that an n-dimensional (or simply nd hereafter) hyperplane can be determined by an $(n-1)$d hyperplane and a point outside of it. We implement our approach as WraLU, and evaluate its performance in terms of precision, efficiency, constraint complexity, and scalability. WraLU outperforms existing advanced methods by generating fewer constraints to achieve tighter approximation in less time. It exhibits versatility by effectively addressing arbitrary input polytopes and higher-dimensional cases, which are beyond the capabilities of existing methods. We integrate WraLU into PRIMA, a state-of-the-art neural network verifier, and apply it to verify large-scale ReLU-based neural networks. Our experimental results demonstrate that WraLU achieves a high efficiency without compromising precision. It reduces the number of constraints that need to be solved by the linear programming solver by up to half, while delivering comparable or even superior results compared to the state-of-the-art verifiers.

CCS Concepts: • **Security and privacy** → **Logic and verification**; • **Computing methodologies** → **Neural networks**.

Additional Key Words and Phrases: Robustness, Neural Networks, Convexity, Polytope

ACM Reference Format:
Zhongkui Ma, Jiaying Li, and Guangdong Bai. 2024. ReLU Hull Approximation. *Proc. ACM Program. Lang.* 8, POPL, Article 75 (January 2024), 29 pages. https://doi.org/10.1145/3632917

1 INTRODUCTION

Over the past decade, neural networks (NNs) have been applied to a variety of real-world applications due to their exceptional performance in solving complex problems. However, like programs, NNs have been found to be vulnerable to security threats and unexpected inputs, posing a great risk to application users. For example, the adversarial perturbation [Szegedy et al. 2014] can cause NNs to make false predictions. Therefore, in addition to testing, it has been widely recognized that NNs should be formally verified.

Verifying the robustness of NNs is a challenging problem though, in part because NNs interleave affine and non-linear activation layers (e.g., ReLU), leading to highly non-linear behaviors. Over the years, researchers have proposed a variety of deterministic methods to address the non-linearity

Authors' addresses: Zhongkui Ma, The University of Queensland, Brisbane, Australia, zhongkui.ma@uq.edu.au; Jiaying Li, Microsoft, Beijing, China, lijiaying1989@gmail.com; Guangdong Bai, The University of Queensland, Brisbane, Australia, g.bai@uq.edu.au.

ACM 2475-1421/2024/1-ART75
https://doi.org/10.1145/3632917

of activation functions for NN verification. Taking ReLU function as an example, its verification methods can be grouped into two categories, i.e., *exact methods* that encode the exact ReLU function using satisfiability modulo theories (SMT) or mixed integer linear programming (MILP) [Bunel et al. 2018; Ehlers 2017; Katz et al. 2017; Ruan et al. 2018; Tjeng et al. 2019], and *approximate methods* that approximate the ReLU function with abstract domains [Gehr et al. 2018; Singh et al. 2018, 2019b], dual approach [Dvijotham et al. 2018b], semidefinite relaxation [Raghunathan et al. 2018], symbolic bound propagation [Weng et al. 2018; Zhang et al. 2022], and convex hull methods [Müller et al. 2022; Singh et al. 2019a]. The exact methods can achieve complete verification but largely suffer from scalability issues in practical scenarios. Therefore, the use of approximate methods has become a common practice.

The core of approximate methods lies in constructing an over-approximation of the activation function [Meng et al. 2022]. Among existing methods, the *convex hull* has been identified as the optimal convex approximation for the activation function [Singh et al. 2019a]. Nonetheless, computing the convex hull itself is a prohibitively expensive task in high-dimensional spaces, and therefore, researchers turn to constructing a *convex polytope* to over-approximate it. Existing research in this area can be divided into two categories. *Single-neuron approximate methods* consider the constraints between a single input and output, and typical examples include Fast-Lin [Weng et al. 2018], CROWN [Zhang et al. 2018], DeepZ [Singh et al. 2018] and DeepPoly [Singh et al. 2019b]. *Multi-neuron approximate methods*, such as k-relu [Singh et al. 2019a], PRIMA [Müller et al. 2022] and OptC2V [Tjandraatmadja et al. 2020], capture the dependency among multiple inputs and outputs to enhance precision. Despite the progress made by all these approaches, achieving a balance between efficiency and precision remains a challenging task.

Our work. In this work, we study the multi-neuron approximation of the *ReLU hull* (denoted by M), which is the convex hull of the ReLU function (denoted by $ReLU$), one of the most used activation function in existing NNs. We propose WraLU[1], a novel, efficient and precise method for approximating ReLU hulls. WraLU reduces the ReLU hull approximation into the problem of computing a convex polytope $\widetilde{M}$, which over-approximates the ReLU hull of the union (X, Y) of a given polytope X and its transformation under the ReLU function $Y = \text{ReLU}(X)$. Its key insight is to leverage the characteristics of the ReLU function as a piece-wise linear function. In particular, the *lower faces* of the ReLU hull have been formulated by combining all linear pieces of the ReLU function, and the *upper faces* can be formulated by constructing another set of faces, each of which is determined by an edge of a lower face and a vertex on the lower faces. Given a polytope X in H-representation, WraLU takes the linear pieces of the ReLU function as the lower faces of $\widetilde{M}$, and finds their edges. It then uses the double description algorithm [Fukuda 2003] to derive the vertices of the lower faces. By determining supporting hyperplanes using each of edge-vertex pairs, WraLU constructs the upper faces of $\widetilde{M}$ (detailed in Sec. 4).

The proposed approach greatly contributes to the balance between efficiency and precision. On efficiency, WraLU includes into $\widetilde{M}$ the exact lower faces of the ReLU hull, which can be directly obtained from the linear pieces of the ReLU function. With the edge of a lower face, only one vertex needs to be figured out to determine a new upper face, speeding up the construction of the upper faces of $\widetilde{M}$. This also maintains a stable number of upper faces, equal to the number of edges of lower faces. In contrast, existing methods often generate a varying number of faces proportional to the dimension and geometry of the convex hull, leading to a potentially large number of faces (and in turn a large number of constraints to solve in the verification). On precision, the inclusion of the exact lower faces ensures the lower bounds of the ReLU hull are included in $\widetilde{M}$, and the upper faces are tight as each of them is determined by at least one edge and one vertex of the lower faces.

[1]WraLU stands for **Wra**pping Re**LU**, resembling constructing a tight approximation of the ReLU hull.

We conduct both intrinsic and extrinsic studies to evaluate the efficacy of WraLU. We first investigate its precision, efficiency, constraints complexity, and scalability by applying it to approximating ReLU hulls with randomly generated input polytopes. On the input polytopes ranging from 2d to 4d, we compare WraLU with three state-of-the-art methods, including the exact method [Singh et al. 2019a], and two approximate methods, i.e., triangle relaxation [Katz et al. 2017] and SBLM+PDDM [Müller et al. 2022]. The results demonstrate that it yields an approximation close to the exact method, achieving a much higher level of precision than triangle relaxation (0.05X–0.2X) and SBLM+PDDM (0.4X–0.7X). WraLU achieves such a high precision with much higher efficiency (10X–10^6X faster than the exact method and up to 30X faster than SBLM+PDDM). Notably, WraLU generates concise approximations with significantly fewer constraints (reducing the number of constraints by up to 99% for the exact method and up to 30% for SBLM+PDDM), reducing the computation burden when using a linear programming solver for subsequent verification. Furthermore, WraLU can generalize to arbitrary input polytopes and remains scalable to dimensions up to 8 (within 10 seconds) that are beyond the capability of all state-of-the-art methods.

We then integrate WraLU into PRIMA, the state-of-the-art multi-neuron verifier, to evaluate its capability in verifying the local robustness of NNs. We measure two metrics, i.e., the number of samples WraLU can verify and the time consumed for verification. Our evaluation utilizes 10 representative benchmarks from MNIST and CIFAR10 datasets provided by ERAN [era 2022]. To investigate the practicality of WraLU, we create another 12 larger-scale NNs trained over a wider range of datasets, including MNIST, CIFAR10, Fashion-MNIST, and EMNIST, with more challenging classification tasks. These benchmarks cover both fully-connected and convolutional NNs. Our results show that WraLU can verify a comparable number of samples to PRIMA but with higher efficiency (up to 50% reduction of verification time). We further examine the factors contributing to the reduction, and find that WraLU generates much fewer constraints within half of the time PRIMA spends, and subsequently reduces the constraint solving time by up to half.
Contributions. This paper makes the following contributions.

- We focus on the ReLU hull approximation problem and study the geometry characteristics of the ReLU hull. Based on this, we propose a new perspective of ReLU hull approximation through identifying the upper faces based on the lower faces derived from the piece-wise ReLU function.
- We develop WraLU, a novel, efficient, and precise method based on locating adjacent faces through incrementally considering output coordinates to approximate the ReLU hull.
- We implement WraLU and evaluated its precision, efficiency, constraints complexity, and scalability. It is also integrated into PRIMA and applied to verify large-scale ReLU NNs.

Notation. Throughout this paper, we use $a, b, d, s, t, x, y, \beta, \lambda$ to denote scalar variables, $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{x}, \boldsymbol{y}$ to denote vectors, and $\boldsymbol{A}$ to denote a matrix. $\boldsymbol{A}_j$ is the j-th row vector of $\boldsymbol{A}$ and b_j is the j-th element of $\boldsymbol{b}$. F, I, L, M, P, Q, X, Y denote polytopes or sets. We denote the set $\{1, 2, \cdots, n\}$ as $[n]$ for simplicity. For convenience, we reuse set intersect operator $\cap$ to denote constraint combining operation. For example, if $X = \{(x_1, x_2) | x_1 > 0, x_2 < 1\}$, then $X \cap \{(x_1, x_2) | x_1 + x_2 > 1\} = \{(x_1, x_2) | x_1 > 0, x_2 < 1, x_1 + x_2 > 1\}$.

2 PROBLEM DEFINITION AND APPROACH OVERVIEW

To ease understanding, we present an overview of WraLU before diving into the details. We present the computational geometry preliminaries on convex polytopes (Sec. 2.1), a definition of the ReLU hull (Sec. 2.2), and illustrate our method of approximating the ReLU hull with a running example (Sec. 2.3).

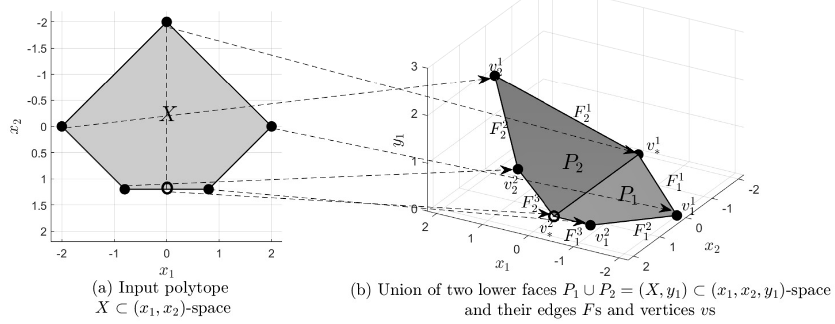

(a) Input polytope $X \subset (x_1, x_2)$-space
(b) Union of two lower faces $P_1 \cup P_2 = (X, y_1) \subset (x_1, x_2, y_1)$-space and their edges Fs and vertices vs

Fig. 1. The polytope X in (x_1, x_2)-space and the union of two lower faces (X, y_1) in (x_1, x_2, y_1)-space under ReLU function $y_1 = \text{ReLU}(x_1)$. The dotted arrows stand for the ReLU transformation.

2.1 Preliminaries of Convex Polytopes

A convex polytope can be represented by either its H- or V-representation. In the literature, the H-representation is commonly used to verify the neural network by linear programming, while the V-representation is essential for constructing ReLU hull approximation. We give the definitions of these two representations below.

Definition (H-representation of polytope). A polytope can be represented by a set $X \subseteq \mathbb{R}^n$ defined by a system of linear constraints $X = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \geq 0\}$, where $\boldsymbol{A} \in \mathbb{R}^{mn}$ and $\boldsymbol{b} \in \mathbb{R}^n$.

Definition (V-representation of polytope). A polytope can be represented by a set $X \subseteq \mathbb{R}^n$ defined by a set of points $V \subset \mathbb{R}^n$, called vertices of X, such that $X = \{\sum_{i=1}^{m} \lambda_i \boldsymbol{v}_i \mid \boldsymbol{v}_i \in V, \sum_i \lambda_i = 1, \lambda_i \in \mathbb{R}, i \in [m]\}$, where m is the number of vertices.

When presenting WraLU's approximation process, the k-face is referred to for describing our algorithm in high dimensional spaces (Sec. 4.1). Therefore, we give its definition below.

Definition (k-face). A k-face ($k \in [n]$) F of a nd polytope $X \subseteq \mathbb{R}^n$ is a kd subset $F \subseteq X$ satisfying $n - k$ linearly independent constraints with equality.

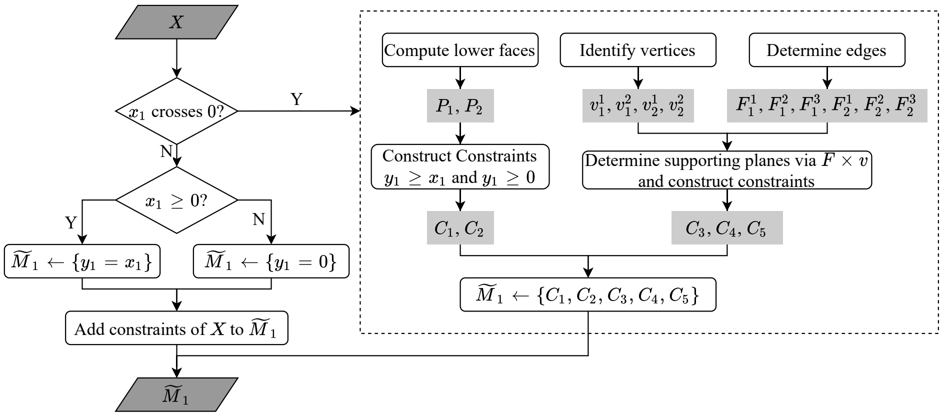
2.2 ReLU Hull and a Running Example

Given a set of points, the *convex hull* is the minimal convex polytope that contains all these points. We give the definition of the convex hull for ReLU below.

Definition (ReLU Hull). Given a bounded convex polytope $X \in \mathbb{R}^n$ of $(x_1, x_2, \cdots, x_n)$ as the domain and its image $Y = \text{ReLU}(X) \in \mathbb{R}^n$ of $(y_1, y_2, \cdots, y_n) = (\text{ReLU}(x_1), \text{ReLU}(x_2), \cdots, \text{ReLU}(x_n))$ under the ReLU function, the convex hull $M \in \mathbb{R}^{2n}$ of the graph (X, Y)[2] is called ReLU hull.

We assume the polytope X is given in its H-representation. Fig. 1 shows a 2d polytope as a running example. In it, $X = \{(x_1, x_2) \mid x_1 + x_2 \geq -2,\ x_1 - x_2 \geq -2,\ -x_1 + x_2 \geq -2,\ -x_1 - x_2 \geq -2,\ -x_2 \geq -1.2\}$, shown in Fig. 1-a), and $Y = \{(y_1, y_2) \mid (x_1, x_2) \in X, y_1 = \text{ReLU}(x_1), y_2 = \text{ReLU}(x_2)\}$ whose y_1 dimension is shown in Fig. 1-b). The goal of WraLU is to obtain an approximation $\widetilde{M}$ to the ReLU hull M of the input polytope X in (x_1, x_2, y_1, y_2)-space, i.e., $\widetilde{M} \supseteq M \supseteq (X, Y)$.

[2] $(X, Y) = (X, \text{ReLU}(X))$ denotes the graph of the ReLU function, i.e., the set of all the points $(x_1, \cdots, x_n, \text{ReLU}(x_1), \cdots, \text{ReLU}(x_n))$, where $(x_1, x_2, \cdots, x_n) \in X$.



X
x_1 crosses 0?
Y
N
$x_1 \geq 0$?
Y
N
$\widetilde{M}_1 \leftarrow \{y_1 = x_1\}$
$\widetilde{M}_1 \leftarrow \{y_1 = 0\}$
Add constraints of X to $\widetilde{M}_1$
$\widetilde{M}_1$
Compute lower faces
P_1, P_2
Construct Constraints $y_1 \geq x_1$ and $y_1 \geq 0$
C_1, C_2
Identify vertices
$v_1^1, v_1^2, v_2^1, v_2^2$
Determine edges
$F_1^1, F_1^2, F_1^3, F_2^1, F_2^2, F_2^3$
Determine supporting planes via $F \times v$ and construct constraints
C_3, C_4, C_5
$\widetilde{M}_1 \leftarrow \{C_1, C_2, C_3, C_4, C_5\}$

Fig. 2. The overall workflow of WraLU on the running example

WraLU incrementally constructs the ReLU hull approximation for a given input polytope by considering one output dimension at a time. It first takes one output dimension and constructs an approximation for the convex hull of the input together with that dimension. The approximation is later used as the input for the next iteration when considering another output dimension. Below we explain this further using our running example.

2.3 Approach Illustration with the Running Example

Without losing generality, WraLU first chooses y_1 and constructs an approximation, denoted by $\widetilde{M}_1$, for the convex hull of $(X, y_1) \subset (x_1, x_2, y_1)$-space, as shown in Fig. 1-b). Taking $\widetilde{M}_1$ as the input, WraLU then constructs an approximation, denoted by $\widetilde{M}_{1,2}$, for the convex hull of $(X, Y) \subset (x_1, x_2, y_1, y_2)$-space. This new approximation $\widetilde{M}_{1,2}$ serves as the final approximation for M. Since the procedure in obtaining $\widetilde{M}_{1,2}$ is similar to $\widetilde{M}_1$, we only discuss the detail in constructing $\widetilde{M}_1$ in the remaining of this section.

To construct $\widetilde{M}_1$, WraLU starts with the lower faces, P_1 and P_2 in Fig. 1-b), which are derived by X and two linear pieces of $y_1 = \text{ReLU}(x_1)$, and utilizes their edges Fs and vertices vs to construct the supporting plane where the faces of $\widetilde{M}_1$ locate. It obtains a polytope formulated by the constraints of these lower faces and the supporting planes and uses it as the convex approximation $\widetilde{M}_1$ to M_1. The main procedure of our method consists of five steps, as shown in Fig. 2. Below we use the running example to illustrate each step.

2.3.1 *Step 1: Computing two lower faces.*

ReLU is a piece-wise linear function which is separated into linear pieces by the non-differentiable points which have $x_i = 0$ in any dimension i. Back to the example, when applying ReLU to x_1, the resulting polyhedron indeed consists of two parts due to the two pieces of ReLU (in Fig. 1-b):

$$P_1 : X \cap \{(x_1, x_2, y_1) \mid x_1 \leq 0, y_1 = 0\}, \qquad P_2 : X \cap \{(x_1, x_2, y_1) \mid x_1 \geq 0, y_1 = x_1\}.$$

Given the convexity of ReLU, P_1 and P_2 are the lower faces of M_1 (detailed proof is in Sec. 3.2). Therefore, P_1 and P_2 are kept as the lower faces of $\widetilde{M}_1$.

2.3.2 *Step 2: Determining open edges of lower faces.*

In this step, our goal is to pinpoint the edges that P_1 and P_2 share with potential upper faces. Given the fact that every edge must be shared and only shared by two faces, there are two types of edges existing, i.e., the edges shared by P_1 and

P_2 themselves (referred to as *closed edges*), and the edges shared by P_1 or P_2 with some unknown faces (referred to as *open edges*), which are the upper faces to be found.

These open edges are transformed from the edges of X by the linear pieces of the ReLU function. Therefore, we have open edges of P_1:[3]

$$F_1^1 = P_1 \cap \{(x_1, x_2, y_1) \mid x_1 + x_2 = -2\},\ F_1^2 = P_1 \cap \{(x_1, x_2, y_1) \mid x_1 - x_2 = -2\},\ F_1^3 = P_1 \cap \{(x_1, x_2, y_1) \mid x_2 = 1.2, \},$$

and those open edges of P_2 are

$$F_2^1 = P_2 \cap \{(x_1, x_2, y_1) \mid x_1 - x_2 = 2\},\ F_2^2 = P_2 \cap \{(x_1, x_2, y_1) \mid x_1 + x_2 = 2\},\ F_2^3 = P_2 \cap \{(x_1, x_2, y_1) \mid x_2 = 1.2\}.$$

2.3.3 *Step 3: Identifying vertices of ReLU Hull.*

In this step, we proceed to pinpoint the vertices of the ReLU hull M_1, taking P_1 and P_2 as inputs. There are two types of vertices on the boundary of X (proved in Sec. 3.3), i.e., (i) those that are transformed from vertices of X on the open edges (the solid vertices in Fig. 1-b), and (ii) those that are introduced by the non-differentiable points (i.e., (0, 0, *) in our running example) on the closed edges (the hollow vertex in Fig. 1-b).

For (i), we apply the double description algorithm [Fukuda 2003] on the H-representation of X, which gives $(-2, 0)$, $(-0.8, 1.2)$, $(2, 0)$, $(0.8, 1.2)$ and $(0, -2)$. They are applied with the ReLU function, resulting in the transformed points in the (x_1, x_2, y_1)-space as follows,

$$v_1^1 = (-2, 0, 0), \qquad v_1^2 = (-0.8, 1.2, 0), \qquad v_2^1 = (2, 0, 2), \qquad v_2^2 = (0.8, 1.2, 0.8),$$

where $v_1^1, v_1^2 \in P_1$ and $v_2^1, v_2^2 \in P_2$. For vertices of (ii), we get one newly-introduced vertex $(0, 1.2, 0)$ by transforming the intersection of X and non-differentiable point of ReLU function to (x_1, x_2, y_1)-space.

2.3.4 *Step 4: Pinpointing supporting planes of faces.*

After Step 3, we have obtained two lower faces, six open edges and one closed edge, four vertices that are only on open edges and two vertices on closed edges. Based on them, this step determines the supporting planes of the faces of $\widetilde{M}_1$, using which the constraints specified by the upper faces can be derived. To this end, we enumerate each combination of an open edge and a vertex, denoted by $L = F \times v$. That is, one supporting plane L is determined by one open edge F and one vertex v outside of F. We note that here we derive the supporting plane rather than the precise face, as computing the boundaries of the face is expensive and we just need the constraints derived by the supporting plane to formulate the H-representation of $\widetilde{M}_1$ (see Step 5).

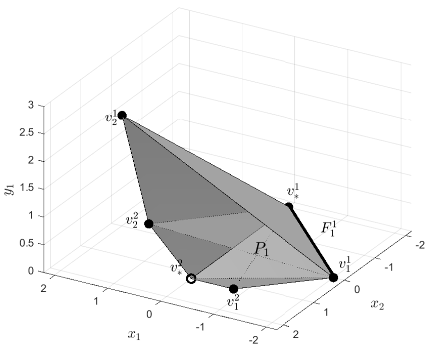


Fig. 3. An illustration of Step 4 using F_1^1 as an example: the plane L_3 is determined by F_1^1 and v_2^1 from four candidate planes

In our running example, Step 4 obtains 24 candidate planes, i.e. $F_1^s \times v_2^t$ and $F_2^s \times v_1^t$ where $s \in [3]$ and $t \in [4]$, where only nine of them are unique. From them, we filter out the planes that are not valid supporting planes, i.e., those that split P_1 or P_2 into two halves. In this way, five planes are kept, where two of them indicate P_1 and P_2, i.e.,

$$L_1 : y_1 = 0, \qquad\qquad L_2 : y_1 = x_1.$$

[3]While there are five candidate groups of constraints generated from the five constraints of X, two infeasible ones, i.e., $-x_1 + x_2 \geq -2$ and $-x_1 - x_2 \geq -2$, which lead to $\emptyset$, are excluded.

The other three, i.e.,

$$L_3 : 2 + x_1 + x_2 = 2y_1, \qquad L_4 : 2 + x_1 - x_2 = 2y_1, \qquad L_5 : -1.2 + x_2 = 0,$$

indicate the planes of three upper faces, denoted by P_3, P_4, and P_5. Fig. 3 takes an open edge F_1^1 as an example to illustrate this process. The plane determined by F_1^1 and v_2^1 is a valid supporting plane, as all vertices lie at one side of it. In contrast, the plane determined by F_1^1 and v_2^2 is discarded, as it splits P_2.

2.3.5 Step 5: Constructing constraints using supporting planes. Taking as input the five supporting planes, L_1, L_2, L_3, L_4, and L_5 found by Step 4, this step converts them into linear constraints. This is achieved by ensuring that all vertices satisfy them, and the following constraints are obtained.

$$C_1 : y_1 \geq 0, \quad C_2 : y_1 \geq x_1, \quad C_3 : 2 + x_1 + x_2 \geq 2y_1, \quad C_4 : 2 + x_1 - x_2 \geq 2y_1, \quad C_5 : 1.2 - x_2 \geq 0.$$

Then, we take it as an approximation to M_1, denoted by $\widetilde{M}_1 = \{(x_1, x_2, y_1) \mid C_1, C_2, C_3, C_4, C_5\}$.

With these five steps, our method can obtain an approximation $\widetilde{M}_1 \supseteq M_1 \supseteq (X, y_1)$. The overall process is iterated with each output coordinate until all are considered.

3 FORMAL FOUNDATION OF OUR APPROACH

In this section, we discuss the intrinsic properties of the ReLU hull. We organize them into five theorems (Theorem 3.1-3.5), which formulate the formal foundation for our proposed approach.

We consider the general cases for the ReLU hull M in the $(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots, y_n)$-space with a given bounded polytope X in the $(x_1, x_2, \cdots, x_n)$-space. Therefore, we assume that all x_is' values $(1 \leq i \leq n)$ cross zero, thereby avoiding trivial scenarios unless explicitly stated otherwise. Whenever such trivial scenarios occur in any dimension, they can be easily handled through linear transformation. This assumption is formalized as follows.

Assumption 1 (Non-trivial Input Polytope). For each dimension $i \in [n]$, there exist two points in X satisfying $x_i > 0$ and $x_i < 0$.

3.1 Linear Pieces of (X, Y)

The first property is about the piece-wise linearity of (X, Y). It has been studied and applied by several previous studies [Bunel et al. 2019; Katz et al. 2017; Wang et al. 2021] and is also leveraged for exact/approximate ReLU hull algorithms [Müller et al. 2022; Singh et al. 2019a].

Theorem 3.1 (Linear Pieces). *$(X, Y) \subset \mathbb{R}^{2n}$ has at most 2^n linear pieces, each of which is*

$$Q_I = X \cap \bigcap_{i \in I} \{(\boldsymbol{x}, \boldsymbol{y}) \mid x_i \leq 0, y_i = 0\} \cap \bigcap_{i \in I^*} \{(\boldsymbol{x}, \boldsymbol{y}) \mid x_i \geq 0, y_i = x_i\}, \tag{1}$$

where $I \subseteq [n]$, $I^* = [n] \setminus I$, and $(\boldsymbol{x}, \boldsymbol{y}) = (x_1, x_2, \cdots, x_n, y_1, y_2, \cdots, y_n)$.

Proof. Each orthant of $\mathbb{R}^{2n}$ space that satisfies $y_i \geq 0$ for any y_i has a linear piece of (X, Y). There are 2^n orthants that satisfy this condition, and (X, Y) therefore has at most 2^n linear pieces. When all x_is $(1 \leq i \leq n)$ cross zero, (X, Y) has the maximum number of linear pieces.

Next, we consider each orthant that has a linear piece, and derive Formula (1). If the orthant takes $x_i \leq 0$ $(i \in I)$, the linear piece has the constraint $y_i = 0$; if the orthant takes $x_i \geq 0$ $(i \in I^*)$, the linear piece has the constraint $y_i = x_i$. These two constraints, together with X, comprise of a linear piece of (X, Y), denoted by Q_I. □

Example. Considering the case of $(X, Y) \subset (x, y)$-space, two linear pieces are $Q_{\{1\}} = X \cap \{(x, y) \mid x \leq 0, y = 0\}$ and $Q_\emptyset = X \cap \{(x, y) \mid x \geq 0, y = x\}$. For the case of $(X, Y) \subset (x_1, x_2, y_1, y_2)$-space, four linear pieces are $Q_{\{1\}} = X \cap \{(x_1, x_2, y_1, y_2) \mid x_1 \leq 0, y_1 = 0, x_2 \geq 0, y_2 = x_2\}$,

$Q_{\{2\}} = X \cap \{(x_1, x_2, y_1, y_2) \mid x_1 \geq 0, y_1 = x_1, x_2 \leq 0, y_2 = 0\}$, $Q_{\{1,2\}} = X \cap \{(x_1, x_2, y_1, y_2) \mid x_1 \leq 0, y_1 = 0, x_2 \leq 0, y_2 = 0\}$ and $Q_\emptyset = X \cap \{(x_1, x_2, y_1, y_2) \mid x_1 \geq 0, y_1 = x_1, x_2 \geq 0, y_2 = x_2\}$.

All properties of the ReLU hull that we discuss in Sec. 3.2 to Sec. 3.4 are centered around the linear pieces of (X, Y). In Sec. 3.2, we show that all these linear pieces are the lower faces of the ReLU hull, and in Sec. 3.3, we prove that all their vertices are exactly all vertices of the ReLU hull.

3.2 Constraints Specified by Lower Faces of ReLU Hull

We first show that those linear pieces derived in Theorem 3.1 are exactly the lower faces of M, and the constraints derived by them give lower bounds of all output variables y_is.

Theorem 3.2 (Lower-Face Constraints). *The ReLU hull M takes all linear pieces in Theorem 3.1 as lower faces, each of which derives one of the following constraints, for $1 \in [n]$,*

$$y_i \geq 0, \qquad y_i \geq x_i. \tag{2}$$

The underlying reason that M has all linear pieces of the ReLU function as its lower faces is the convexity of the ReLU function. With this insight, below we give a proof for the theorem.

Proof. First, we have that $y_i = 0$ and $y_i = x_i$ are supporting hyperplanes of M, because all points of M satisfy $y_i \geq 0$ and $y_i \geq x_i$ by ReLU transformation and there exist points on $y_i = 0$ or $y_i = x_i$.

Second, we prove that $y_i = 0$ and $y_i = x_i$ are faces of M. For each hyperplane $y_i = 0$ or $y_i = x_i$, there exist at least n linearly independent points of M on it, given that each hyperplane contains one linear piece in Theorem 3.1, and this linear piece has at least n such points. □

We remark that these constraints are commonly utilized in approximating the ReLU hull, such as the triangle relaxation [Katz et al. 2017].

Example. Considering our running example, two constraints $y_1 \geq 0$ and $y_1 = x_1$ specified by the two lower faces P_1 and P_2 are shown in Fig. 1.

3.3 Vertices of ReLU Hull

Theorem 3.2 has established that all linear pieces are the lower faces of the ReLU hull. Due to this, their vertices are naturally the vertices of the ReLU hull. Then we have the following theorem.

Theorem 3.3 (ReLU Hull Vertices). *The vertices V_M of M is the union of vertices V_{Q_I} of all linear pieces Q_I in Theorem 3.1, i.e.,*

$$V_M = \bigcup_{I \subseteq [n]} V_{Q_I}. \tag{3}$$

Proof. First, we prove $V_M \supseteq \bigcup_{I \subseteq [n]} V_{Q_I}$. Because M takes each linear piece in Theorem 3.1 as a face, all vertices of these linear pieces are contained in vertices of M.

Second, we prove $V_M \setminus \bigcup_{I \subseteq [n]} V_{Q_I} = \emptyset$. Assuming there exists an extra vertex of M outside these linear pieces, a smaller convex polytope can be formulated by using only the vertices of all linear pieces, leading to a contradiction that M is the convex hull. □

Next, we trace back the vertices V_M to X which is in the input space of the ReLU function, to find those vertices that can be used to determine the upper faces. All vertices of M are either transformed from the vertices of X or introduced by the non-differentiable points of the ReLU function. Specifically, we have the following theorem.

Theorem 3.4 (Sources of ReLU Hull Vertices). *Given a vertex v of M, v is transformed under the ReLU function from one or many of the following three types of points,*

(1) *the vertex of X,*

(2) *the intersection of a coordinate axis and the boundary of X, and*
(3) *the origin.*

Proof. By Theorem 3.3, the vertices V_M of M is the union of vertices V_{Q_I} of all linear piece Q_I, so here we only need to discuss the sources of vertices V_{Q_I}. By Theorem 3.1, such a vertex $v \in V_{Q_I}$ should satisfy $2n$ constraints in three sets of ① $\{\boldsymbol{A}_j\boldsymbol{x} + \boldsymbol{b}_j = 0 \mid j \in [m]\}$, ② $\{y_i = 0 \mid i \in [n]\}$, and ③ $\{y_i = x_i \mid i \in [n]\}$, because v is in the $2nd$ $(x_1, \cdots, x_n, y_1, \cdots, y_n)$-space. Through identifying the sources of these $2n$ constraints, we can find all sources of v.

Because v is of one linear piece, it satisfy n constraints of $y_i = 0$ or $y_i = x_i$ for each $i \in [n]$ from ② and ③ by the definition of linear piece (Formula (1)). Then, we only need to consider the sources of the other n constraints. For source (1), these extra n constraints are all from ① and v is a vertex of X in $(x_1, \cdots, x_n)$-space. For source (2), there are extra m ($m < n$) constraints from ①, and the other $n - m$ constraints are from ② or ③. Because v satisfies m constraints of ①, it is on the boundary of X. On the other hand, for some coordinate i, $y_i = x_i$ and $y_i = 0$ can both hold, so the vertex is on the axis. For source (3), these extra n constraints are all from ② or ③. Then, v satisfies all $2n$ constraints of ② and ③, so v is the origin. □

Example. For our running example, v_*^1 and v_*^2 are vertices from the intersection of axis x_1 and the boundary (in the line $x_2 = 1.2$) of X in (x_1, x_2)-space. Vertices v_2^1, v_2^2, v_1^1, v_1^2, and v_*^2 are from the vertices of X in (x_1, x_2)-space. Note that v_*^2 can be led to from two sources, (1) and (2) in Theorem 3.4. The origin is not a vertex in (x_1, x_2, y_1)-space but a vertex in (x_1, x_2, y_1, y_2)-space. Note that all vertices are on the linear pieces and on the boundary of X except for the origin.

Remark. Among these three types of vertices, WraLU can exclude some for efficiency. First, according to Formula (1), the origin is at the intersection of all low faces. Based on Assumption 1, it must be in the interior of X, and cannot be in any upper face. Otherwise, such an upper face will split M. Therefore, based on Theorem 3.4, all possible vertices that we rely on to identify the upper faces are transformed from a point on the boundary of X excluding the origin, i.e., (1) and (2) in the theorem. In fact, we can further exclude the vertices of (2) as each of them can be replaced by at least one vertex of (1) which is in the same face. We defer the proof of this to Sec. 5.3.

3.4 Constraints Specified by Upper Faces of ReLU Hull

This section discusses the constraints derived by the upper faces of the ReLU hull. They either provide upper bounds to at least one output variable y_i or do not contain any y_i. A constraint without any y_i is from one constraint of the H-representation of X. The following theorem reveals the form of the constraints specified by upper faces.

Theorem 3.5 (Upper-Face Constraints). *Given the H-presentation of X, i.e., $\{(x_1, x_2, \cdots, x_n) \mid \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \geq 0, \boldsymbol{A} \in \mathbb{R}^{mn}, \boldsymbol{b} \in \mathbb{R}^n\}$, all constraints specified by the upper faces of M are of the following form,*

$$\boldsymbol{A}_j\boldsymbol{x} + b_j \geq \sum_{1 \leq i \leq n} \beta_{i1} y_i + \beta_{i2}(y_i - x_i), \tag{4}$$

where $j \in [m]$, $i \in [n]$, and $\beta_{ik} \in \mathbb{R}$ ($k \in [2]$) are constants.

Proof. First, we discuss the general form of the supporting hyperplane specified by a face of M. The key point is that each face contains at least one open edge, and we construct a hyperplane to contain this open edge. Each open edge is the intersection of a lower face and a boundary of X. Each lower face is a linear piece of (X, Y) with Formula (1) and is the intersection of $y_i = 0$ or $y_i = x_i$ for each dimension $i \in [n]$. The boundary of X is one of hyperplane $\boldsymbol{A}_j\boldsymbol{x} + b_j = 0$ by the constraints of X. Therefore, each open edge is the intersection of $y_i = 0$ (or $y_i = x_i$) for any i and

$\boldsymbol{A}_j\boldsymbol{x} + b_j = 0$ for one j. Then, $\beta_{i0}(\boldsymbol{A}_j\boldsymbol{x} + b_j) = \sum_{1\le i\le n} \beta_{i1}y_i + \beta_{i2}(y_i - x_i)$ with parameters β_{i0}, β_{i1}, and β_{i2} represents a hyperplane crossing an open edge.

Second, we give the general form of the supporting hyperplane specified by an upper face of M. We aim to prove that when $\beta_{i0} \ne 0$, the general form implies those supporting hyperplanes specified by upper faces. When $\beta_{i0} = 0$, $\sum_{1\le i\le n} \beta_{i1}y_i + \beta_{i2}(y_i - x_i) = 0$ can imply all supporting hyperplanes $y_i = 0$ and $y_i = x_i$ determined by lower face with $\beta_{i1} = 0$ or $\beta_{i2} = 0$. Additionally, with $\beta_{i1}\beta_{i2} \ne 0$, all supporting hyper planes $\sum_{1\le i\le n} \beta_{i1}y_i + \beta_{i2}(y_i - x_i) = 0$ can be derived by $y_i = 0$ and $y_i = x_i$ and are unnecessary. Therefore, to exclude the unnecessary cases of lower faces, the hyperplane $(\boldsymbol{A}_j\boldsymbol{x} + b_j) = \sum_{1\le i\le n} \beta_{i1}y_i + \beta_{i2}(y_i - x_i)$ without β_{i0} represents the general form of the supporting hyperplanes specified by an upper face.

Last, we derive Formula (4) based on the form of the supporting hyperplane. Considering the origin is in X and is a vertex of M by Theorem 3.4, the origin should satisfy the upper-face constraint. We substituting the origin in the supporting plane and have $\boldsymbol{Ax} + \boldsymbol{b} \ge 0$ by constraints of X and $\beta_{i1}y_i + \beta_{i2}(y_i - x_i) = 0$. Therefore, we take $\ge$, and the constraint determined by the supporting plane is $\boldsymbol{A}_j\boldsymbol{x} + b_j \ge \sum_{1\le i\le n} \beta_{i1}y_i + \beta_{i2}(y_i - x_i)$. □

Example. Considering our running example in Fig. 3, an upper face in the hyperplane L_3 is determined by F_1^1 and v_2^1 and has a form of $2 + x_1 + x_2 = 2y_1$, which contains the open edge F_1^1, i.e., the intersection of $2+x_1+x_2 = 0$ and $y_1 = 0$. The constraints C_3 determined by L_3 is $2+x_1+x_2 \ge 2y_1$. Noted that $2 + x_1 + x_2 \ge 0$ is a constraint of X and $2 + x_1 + x_2 \ge 2y_1$ is consistent with the general form of upper-face constraints in Theorem 3.5.

Note that Formula (4) can be extended to include the constraints of lower faces (Theorem 3.2), by introducing a new parameter β_{i0} indicating $\beta_{i0}(\boldsymbol{A}_j\boldsymbol{x} + b_j)$, so that it can represent a general constraint of the ReLU hull. We highlight that the derived form of the ReLU hull is so far the most general one. The state-of-the-art method for ReLU hull approximation [Tjandraatmadja et al. 2020] produces a similar form, but it entails a premise that the input X must be a hyperrectangle and it considers only one output coordinate.

Finding an optimal combination of the parameters β_{ik} in Formula (4) results in a face of the exact ReLU hull. However, this is a computationally expensive task, because it is well known that the number of constraints typically is an exponential multiple of the dimensions of the variables [McMullen 1970]. Therefore, we propose to determine each of them individually, and each time a vertex is taken to find an β_{ik} by determining the hyperplane which contains an upper face (detailed in Sec. 4). This process does not guarantee to identify the optimal combinations, but retains soundness and tightness of the over-approximation (analyzed in Sec. 5.1).

4 WRALU: A RELU HULL OVER-APPROXIMATION APPROACH

In this section, we detail WraLU for general cases, including the main algorithm (Sec. 4.1), and two innovations (Sec. 4.2), which boosts the efficiency of pinpointing the upper faces.

4.1 The Main Algorithm

Algo. 1 outlines the main process of WraLU without optimization applied. It takes as input a polytope X and an output coordinate order of $1, 2, \cdots, n$, denoted by $Order$, and outputs another polytope $\widetilde{M}$ as a ReLU hull approximation. Both X and $\widetilde{M}$ are in H-representation. $\widetilde{M}$ is constructed in an iterative manner by each output coordinate (the main loop dominated by line 3), and it stores the current approximation of the ReLU hull (in a lower dimension) during the iteration (line 22).

WraLU starts with initializing $\widetilde{M}$ with the input polytope X. During each iteration, it checks whether the ReLU for the current dimension i is a trivial case, i.e., all x_i's values are either non-positive or non-negative. If so, it can construct the exact convex hull in the current dimension (lines

7–8) and finish the current iteration. Otherwise, an approximation is necessary. In fact, our running example is a non-trivial case, and the process below is a generalization of the five steps in Sec. 2.3.

Algorithm 1: WraLU Basic(X, $Order$)

```
1  $Vs \leftarrow GetVertices(X)$;
2  $\widetilde{M} \leftarrow X$;
3  while $i \leftarrow GetNextOutputDimension(Order)$ do
4      $Fs \leftarrow \emptyset$;
5      $n \leftarrow GetDimension(\widetilde{M})$;
       // Transform vertices in new dimension
6      Update $Vs$ by applying ReLU on coordinate $y_i$;
       // Trivial cases
7      if $isNonPositive(i)$ then $\widetilde{M}.add(y_i = 0)$; continue;
8      if $isNonNegative(i)$ then $\widetilde{M}.add(y_i = x_i)$; continue;
       // General cases
9      $\widetilde{M}' \leftarrow \emptyset$;
       // Get two lower faces
10     $P_1 \leftarrow \widetilde{M} \cap \{x_i \leq 0,\ y_i = 0\}$;
11     $P_2 \leftarrow \widetilde{M} \cap \{x_i \geq 0,\ y_i = x_i\}$;
       // Add constraints specified by two lower faces
12     $\widetilde{M}'$.add($\{y_i \geq 0, y_i \geq x_i\}$);
       // Determine edges
13     foreach constraint $\boldsymbol{ax} + \boldsymbol{by} + c \geq 0$ in $\widetilde{M}$ do
           // Get the $(n+i-1)$d hyperplane specified by the constraint
14         $p \leftarrow \boldsymbol{ax} + \boldsymbol{by} + c = 0$;
15         if $P_1 \cap p \neq \emptyset$ then $Fs$.add($p$) ;
16         if $P_2 \cap p \neq \emptyset$ then $Fs$.add($p$) ;
       // Determine supporting hyperplanes and add derived constraints
17     foreach $F$ in $Fs$ do
18         foreach $v$ in $Vs$ do
               // Determine a supporting hyperplane
19             $L \leftarrow F \times v$;
20             if $L$ splits $P_1$ or $P_2$ into halves then continue;
21             $\widetilde{M}'$.add($GetConstraint(L, Vs)$);
       // Update the approximation
22     $\widetilde{M} \leftarrow \widetilde{M}'$
23 return $\widetilde{M}$;
```

WraLU first obtains the constraints $y_i \geq 0$ and $y_i \geq x_i$ derived by the two lower faces P_1 and P_2 (line 12), and then attempts to find additional supporting hyperplanes specified by upper faces (lines 13–21). To do this, it determines the $(n+i-2)$-faces (edges in general high-dimensional space) and vertices, and leverages them to obtain the $(n+i-1)$d supporting planes in which the upper faces are located. Specifically, we only need the supporting planes of these $(n+i-2)$-faces, and they are stored in the set Fs that is calculated based on $\widetilde{M}$ in the previous iteration (lines 13–16). Note that WraLU checks whether $(n+i-2)$-face p is an open edge by checking $P_1 \cap p \neq \emptyset$ and $P_2 \cap p \neq \emptyset$ (lines 15 and 16). A *lazy selection* is introduced to avoid this costly operation (Sec. 4.2.1). The vertices are obtained by applying the ReLU transformer to the vertices of the input polytope (line 6), which is handled in an iterative manner (Sec. 4.2.2). The rationale of this step is discussed in Sec. 4.2.2, and its correctness is proved in Sec. 5.3. WraLU then updates the approximation $\widetilde{M}$ by $\widetilde{M}'$ which contains the constraints that do not split P_1 or P_2 into halves, i.e., these constraints represent supporting planes (lines 20 and 21). The iteration terminates after every output dimension y_i has been enumerated, and WraLU outputs a sound over-approximation $\widetilde{M}$ of the ReLU hull (X, Y).

Complexity. Algo. 1 takes as input the double description, i.e., the constraints and vertices, of a d-dimensional polytope, which is a premise consistent with PRIMA [Müller et al. 2022]. Given n_a constraints and n_v vertices, WraLU has a time complexity of $O(d^2 n_a n_v)$. First, lines 13–16 in Algo. 1 loop all constraints, which has a complexity of $O(n_a)$. Second, lines 17–21 iterate all pairs of edges and vertices ($O(n_a n_v)$) to calculate β ($O(d)$) in line 20, so lines 17–21 have a complexity of $O(dn_a n_v)$. Therefore, the overall time complexity is $O(d^2 n_a n_v)$.

4.2 Two Innovations for Efficiency

WraLU greatly leverages the characteristics of the ReLU hull for its efficiency. Besides using the linear pieces as the lower faces (lines 10–11), it is worth highlighting that two innovations regarding the identification of the upper-face constraints for efficiency.

4.2.1 *Innovation #1: Lazy Selection with Incremental Calculation of β_{ik}s.*

Algo. 1 involves three condition checkings that are costly, and WRALU addresses this by introducing a *lazy selection* based on β_{ik}s (in Formula (4). First, to determine the conditions $P_1 \cap p \neq \emptyset$ and $P_2 \cap p \neq \emptyset$ (lines 15 and 16), a naive but costly way is to calculate the exact P_1 and P_2. Although the H-representation of P_1 and P_2 can easily obtained by the intersection of $\widetilde{M}$ with the ReLU constraints $(x > 0) \wedge (y = x)$ or $(x < 0) \wedge (y = 0)$, checking $P_1 \cap p \neq \emptyset$ and $P_2 \cap p \neq \emptyset$ is costly due to the redundancy in the intersection. Instead, WRALU skips the condition checking and adds both potential open edges into Fs. This causes two hyperplanes being constructed in line 19. One of them is not tight enough, and it leads to constraints with a $\beta_{ik} = 0$, while the tighter one has a positive β_{ik}. Based on this, WRALU conducts filtering after line 19.

Second, checking the condition whether L splits P_1 or P_2 into halves (line 20) may entail checking whether all vertices of M are in the same side of L. This can be costly too, as to do this, WRALU has to calculate all vertices. To alleviate it, for each open edge F in Fs (line 17), WRALU calculates all planes that are determined by F and each of the vertex in Vs. This leads to $|Vs|$ constraints of the form in Formula (4). From them, WRALU keeps only the constraint with the least β_{ik} in line 21.

Workflow. Innovation #1 is based on (1) identifying the value of β_{ik} that provides a sound constraint (to be formulated by Theorem 5.2), and (2) filtering out the constraints with any non-existing open edge indicating by zero value of β_{ik} (to be formulated by Theorem 5.3).

The main idea is to construct a parametric function with one parameter β_{ik} for the current y_i to represent a hyperplane that crosses a potential open edge. WRALU valuates this parameter using every vertex (line 19), resulting in several candidate hyperplanes with different values of β_{ik}. After that, it selects the one with an β_{ik} value that satisfies the conditions in lines 15, 16, and 20 to be the upper face. Below, we use the case of $i = 1$ to demonstrate the procedure and brief its soundness. The procedure and analysis are directly applicable to other iterations. For conciseness, the parameters β_1 and β_2 omit the subscripts that indicate the dimension.

First, in lines 15 and 16, WRALU skips the checking of the two conditions, causing two potential open edges being added into the edge set Fs. These two edges later are used for constructing hyperplanes, as represented by the following two parametric functions,

$$\boldsymbol{A}_j \boldsymbol{x} + b_j = \beta_1 y_1, \qquad \boldsymbol{A}_j \boldsymbol{x} + b_j = \beta_2 (y_1 - x_1). \tag{5}$$

Then, a process of determining β_1 and β_2 is introduced to replace the operation in line 20, and achieves its purpose of filtering out a L that does not split P_1 and P_2 into halves. $\boldsymbol{A}_j$ and b_j are specified by the j-th constraint of X. WRALU substitutes $\boldsymbol{x}$ and y_1 with each vertices. Those vertices with $y_1 \neq 0$ will give all candidate values of β_1, and those vertices with $y_1 \neq x_1$ will be used to identify all candidate values of β_2 by the following equations,

$$\beta_1 = \frac{\boldsymbol{A}_j \boldsymbol{x} + b_j}{y_1}, \qquad \beta_2 = \frac{\boldsymbol{A}_j \boldsymbol{x} + b_j}{y_1 - x_1}. \tag{6}$$

β_1 and β_2 take the least value of their candidate values, respectively, to generate a sound constraint (to be proved in Theorem 5.2).

With the values of β_1 and β_2 determined, we can obtain two sound constraints derived by Formula (6) as follows (formulated in Theorem 5.2),

$$\boldsymbol{A}_j \boldsymbol{x} + b_j \geq \beta_1 y_1, \qquad \boldsymbol{A}_j \boldsymbol{x} + b_j \geq \beta_2 (y_1 - x_1). \tag{7}$$

Type 1 redundant constraints. One of the two constraint candidates in Formula (7) is redundant, as it is not as tight as the other. We refer to this type of redundant constraints as *Type 1*, to distinguish them from another type resulted from the redundant constraints of the input polytopes (to be discussed soon in this section). Such constraints can be omitted without affecting the definition of the polytope. The lazy filtering of WraLU is conducted to filter out Type 1 redundant constraints, by considering the following two cases.

- If both of β_1 and β_2 are zero, the two constraints are identical and we add the constraint $\boldsymbol{A}_j\boldsymbol{x} + b_j \geq 0$ in $\widetilde{M}'$ (line 21).
- If one is zero and one is positive, the parametric function with the former indicates a non-existing edge (discussed in Sec. 5.3). Therefore, we select the constraint with positive β and add it in $\widetilde{M}'$ (line 21).

Both cases lead to one of the constraints in Formula (7) being selected, and thus this process is equivalent to operating lines 15 and 16 (to be proved in Theorem 5.3). We highlight that during the iteration, the number of upper-face constraints is stable, because WraLU replaces the original constraints with one of the two constraint candidates. This results in the number of constructed constraints being the same as that of the constraints of the input polytope X, if not counting the lower-face constraints.

Type 2 redundant constraints. Another type of redundant constraints stems from the input polytope whose constraints are generated by DeepPoly [Singh et al. 2019b]. We refer to them as *Type 2* redundant constraints. The Type 2 redundant constraint $\boldsymbol{A}_j\boldsymbol{x} + \boldsymbol{b}_j \geq 0$ causes β_1 and β_2 both positive, which is different from the non-redundant constraints of X (see the two cases in Type 1 handling). This makes it infeasible to determine which of these two constraints is tighter, so WraLU keeps the constraint $\boldsymbol{A}_j\boldsymbol{x} + b_j \geq \beta_{i1}y_1 + \beta_{i2}(y_1 - x_1)$ rather than both of them. The rationale is because vertices of M are on lower faces and cannot take both of y_i and $y_i - x_i$ being non-zero. Specifically, those vertices with $y_1 = x_1$ satisfy $\boldsymbol{A}_j\boldsymbol{x} + b_j \geq \beta_{i1}y_1$ where $\beta_{i2}(y_1 - x_1) = 0$, and those with $y_1 = 0$ satisfy $\boldsymbol{A}_j\boldsymbol{x} + b_j \geq \beta_{i2}(y_1 - x_1)$ where $\beta_{i1}y_1 = 0$. Therefore, the construction of the constraint $\boldsymbol{A}_j\boldsymbol{x} + b_j \geq \beta_{i1}y_1 + \beta_{i2}(y_1 - x_1)$ retains the soundness.

Example. We use our running example to show the process of determining the constraint C_3. Firstly, we choose the constraint $2 + x_1 + x_2 \geq 0$ of X and set two constraint candidates $2 + x_1 + x_2 \geq \beta_1 y_1$ and $2 + x_1 + x_2 \geq \beta_2(y_1 - x_1)$. The former candidate contains the open edge $F_1^1 = \{(x_1, x_2, y_1) \mid 2 + x_1 + x_2, y_1 = 0\}$, but the latter candidate contains a non-existing open edge $\{(x_1, x_2, y_1) \mid 2 + x_1 + x_2, y_1 = x_1\}$. WraLU tests all vertices satisfying $y_1 \neq 0$ to calculate β_1, and identifies $v_2^1 = (2, 0, 2)$ determining $\beta_1 = 2$, which provides a sound constraint. Similarly, we obtain $\beta_2 = 0$ determined by v_1^1. Then, we discard the constraint containing a non-existing open edge with $\beta_2 = 0$ and keep the constraint $2 + x_1 + x_2 \geq \beta_1 y_1$ with $\beta_1 = 2$. Additionally, $2 + x_1 + x_2 \geq \beta_1 y_1$ provides a upper bound of y_1 due to positive β_1 (in Fig. 3).

4.2.2 Innovation #2: Iterative Vertices Update. In each iteration, the vertices of the $\widetilde{M}$ from the previous iteration are required to determine the upper faces (lines 18–21). A naive way is to calculate these vertices in each iteration, but this is costly due to the exponential complexity of converting H-representation to V-presentation [Fukuda and Prodon 1995]. We propose to calculate the vertices of X once (i.e., $GetVertices$ in line 1), and update them in each iteration (line 6). The update is conducted by extending the coordinates of these vertices in i-th output dimension by $y_i = \text{ReLU}(x_i)$. The correctness of this is proved in Sec. 5.3.

Example. For the example in Sec. 2, the vertex v_*^2 is not transformed from the vertex of X and is generated by the non-differential points with $y_1 = x_1$ of $y_1 = \text{ReLU}(x_1)$. Specifically, it cannot determine an upper face by calculating β (Formula (6)) because it always makes $y_1 = 0$ and $y_1 = x_1$.

5 SOUNDNESS OF WRALU

In this section, we prove the soundness of WraLU. To ease the understanding, we first establish the soundness of the main algorithm without considering the two innovations (Sec. 5.1). Based on that, we show that introducing the two innovations retains the soundness (Sec. 5.2 and Sec. 5.3).

5.1 Soundness of the Main Algorithm

First, we demonstrate the soundness of Algo. 1. That is, the ReLU hull must be contained by the generated approximation.

THEOREM 5.1 (SOUNDNESS OF MAIN ALGORITHM). *The polytope $\widetilde{M}$ produced by Algo. 1 is an over-approximation to the ReLU hull M of the given input polytope X, i.e., $\widetilde{M} \supseteq M$.*

PROOF. We prove the theorem by induction. To distinguish between iterations, we use $\widetilde{M}_i$ to denote $\widetilde{M}$ in Algo. 1 after the i-th iteration, with $\widetilde{M}_0 = X$. We assume that the order specified by *Order* is $1, 2, 3, \cdots, n$.

For the base case, we have $\widetilde{M}_0 = M_0 = X$.

Given the induction hypothesis $\widetilde{M}_i \supseteq M_i$, we need to prove $\widetilde{M}_{i+1} \supseteq M_{i+1}$ under the ReLU function $y_i = \mathrm{ReLU}(x_i)$ by Algo. 1.

The exact ReLU hull M_{i+1} is the minimal convex polytope satisfying $M_{i+1} \supseteq (M_i, y_{i+1})$. We need to prove that $\widetilde{M}_{i+1} \supseteq (\widetilde{M}_i, y_{i+1}) \supseteq (M_i, y_{i+1})$, and then the convex polytope $\widetilde{M}_{i+1} \supseteq M_{i+1}$.

First, $(\widetilde{M}_i, y_{i+1}) \supseteq (M_i, y_{i+1})$ under $y_{i+1} = \mathrm{ReLU}(x_{i+1})$, because of $\widetilde{M}_i \supseteq M_i$.

Second, to prove $\widetilde{M}_{i+1} \supseteq (\widetilde{M}_i, y_{i+1})$, let P_1 and P_2 be the two pieces of $(\widetilde{M}_i, y_{i+1})$. We prove $\widetilde{M}_{i+1} \supseteq P_1$ and $\widetilde{M}_{i+1} \supseteq P_2$, and then $\widetilde{M}_{i+1} \supseteq (\widetilde{M}_i, y_{i+1}) = P_1 \cup P_2$. To prove $\widetilde{M}_{i+1} \supseteq P_1$, we just need to prove that all vertices of the convex polytope P_1 is in $\widetilde{M}_{i+1}$. Let v be a vertex in P_1. Below we show that v satisfies any constraint C of $\widetilde{M}_{i+1}$ determined by Algo. 1.

The constraint C is specified by either (i) the lower faces, or (ii) the upper faces. For (i), v satisfies $y_i \geq 0$ and $y_i \geq x_i$ because it is transformed by the ReLU function. Regarding (ii), v satisfies them because WRALU keeps only supporting hyperplane candidates that do not divide the vertices (lines 20 in Algo. 1). Therefore, we conclude that v satisfies any constraint C in $\widetilde{M}_{i+1}$ and $v \in \widetilde{M}_{i+1}$, and thus $P_1 \subseteq \widetilde{M}_{i+1}$. Due to symmetricity, $P_2 \subseteq \widetilde{M}_{i+1}$ is proved and $\widetilde{M}_{i+1} \supseteq P_1 \cup P_2$.

Hence, we have $\widetilde{M}_{i+1} \supseteq M_{i+1}$, concluding the theorem. □

5.2 Correctness of Innovation #1

In this section, we demonstrate that the Innovation #1, i.e., lazy selection with the incremental calculation of β_{ik}, can yield an equivalent effect as the operations in lines 15, 16, and 21 of Algo. 1. That is, Innovation #1 retains the soundness of the main algorithm. For conciseness, we present the analysis on the case of dimension $i = 1$, while the general case is given at the end of this section.

5.2.1 Constraints Identification (line 20). Recall that to derive the constraints, β_1 or β_2 takes the minimum value of all candidate values determined by vertices. Theorem 5.2 ensures that the constraints derived are implied by the ReLU hull.

THEOREM 5.2 (EQUIVALENCE TO LINE 20). *For one formula $\boldsymbol{A}_j\boldsymbol{x} + b_j$ specified by X, the generated constraint $\boldsymbol{A}_j\boldsymbol{x} + b_j \geq \beta_1 y_1$ or $\boldsymbol{A}_j\boldsymbol{x} + b_j \geq \beta_2(y_1 - x_1)$ is sound, when*

$$\beta_1 = \min\left\{\frac{\boldsymbol{A}_j\boldsymbol{x}' + b_j}{y_1'}\right\}, \qquad \beta_2 = \min\left\{\frac{\boldsymbol{A}_j\boldsymbol{x}' + b_j}{y_1' - x_1'}\right\}, \tag{8}$$

where $(\boldsymbol{x}', y')$ is a vertex of M satisfying $y_1' \neq 0$ or $y_1' \neq x_1'$.

PROOF. Due to symmetricity, we only prove the case of β_1. We prove this theorem by contradiction. We assume that β_1 does not take the minimal value, and leads to a contradiction. Let $v'' = (\boldsymbol{x}'', y_1'')$ with $y_1'' \neq 0$ be the vertex making β_1 minimal and $v' = (\boldsymbol{x}', y_1')$ makes $\beta_1 = \frac{\boldsymbol{A}_j\boldsymbol{x}'+b_j}{y_1'}$. Because $\frac{\boldsymbol{A}_j\boldsymbol{x}''+b_j}{y_1''} < \beta_1 = \frac{\boldsymbol{A}_j\boldsymbol{x}'+b_j}{y_1'}$, we will have $\boldsymbol{A}_j\boldsymbol{x}'' + b_j < \beta_1 y_1''$. Then v'' does not satisfy the generated constraints $\boldsymbol{A}_j\boldsymbol{x} + b_j \geq \beta_1 y_1$. Hence, $\boldsymbol{A}_j\boldsymbol{x} + b_j \geq \beta_1 y_1$ is not sound. □

5.2.2 Open Edges Filtering (lines 15 and 16). Recall that WRALU uses a further selection between β_1 and β_2 to filter out non-existing open edges through comparing the values of β_1 and β_2. As a result, it only keeps one constraint from Formula (7). Theorem 5.3 establishes its soundness.

THEOREM 5.3 (EQUIVALENCE TO LINES 15 AND 16). *Given $\boldsymbol{A}_j\boldsymbol{x} + b_j$ that is specified by a non-redundant constraint of X, β_1 and β_2 in Formula (7) satisfy one of the following two cases,*

(1) *$\beta_1 = \beta_2 = 0$, or*
(2) *$\beta_1\beta_2 = 0$ and one of them is positive,*

PROOF. We prove this by contradiction. We assume that $\beta_1 = \min\left\{\frac{\boldsymbol{A}_j\boldsymbol{x}+b_j}{y_1}\right\} \neq 0$ and $\beta_2 = \min\left\{\frac{\boldsymbol{A}_j\boldsymbol{x}+b_j}{y_1-x_1}\right\} \neq 0$. There must be a vertex satisfying $\boldsymbol{A}_j\boldsymbol{x} + b_j = 0$ in one lower face, because $\boldsymbol{A}_j\boldsymbol{x} + b_j$ is specified by a non-redundant constraint of X. This vertex makes $y_1 \neq 0$ or $y_1 - x_1 \neq 0$, which means that β_1 or β_2 can be identified as zero. There is a contradiction. □

By Theorem 5.3, we only keep one constraint of Formula (7) with the positive β if one of β_1 and β_2 is non-zero, because only the non-zero β indicates an existing open edge and a tighter constraint. Otherwise, $\beta_1 = \beta_2 = 0$, and the two constraints are identical. We then keep $\boldsymbol{A}_j\boldsymbol{x}+b_j \geq 0$ into approximation. In fact, when $\beta_1 = \beta_2 = 0$, it means that $P_1 \cap p$ and $P_2 \cap p$ are both open edges. The identification of β_1 and β_2 indicates that there exist two vertices satisfying $y_1 \neq 0$ and $y_1 \neq x_1$, respectively, and they are on the two hyperplanes in Formula (5). Then, each of these two hyperplanes contains a vertex of M, so each of them contains an open edge.

5.2.3 Non-negativeness of β_{ik}. The following theorem gives the non-negativeness of β_k ($k \in [2]$). This indicates that the constraints generated by WRALU always give a lower bound to those y_is with a positive coefficient. This contributes to a tight approximation.

THEOREM 5.4 (NON-NEGATIVENESS OF β_{ik}). *The values of β_1 and β_2 in Formula (6) that are identified by vertices of M satisfy*

$$\beta_1 \geq 0, \qquad \beta_2 \geq 0 \tag{9}$$

PROOF. Due to symmetricity, We only prove the case of β_1. First, all vertices of M satisfy $\boldsymbol{A}_j\boldsymbol{x}+b_j \geq 0$, because such a constraint is of the H-representation of X. Second, all vertices satisfy $y_i \geq 0$. Therefore, the values determined by the vertices with $y_i > 0$ in Formula (6) are positive. □

General Cases. To handle general cases, for each $\boldsymbol{A}_j\boldsymbol{x}+b_j$ specified by the constraints of X, WRALU adds y_i or $y_i - x_i$ items with β_{i1} or β_{i2} in each iteration based on the general form in Theorem 3.5. It determines the values of β_{i1} and β_{i2} based on the following two parametric functions,

$$\boldsymbol{A}_j\boldsymbol{x}+b_j - \sum_{h=0}^{i-1}[\beta_{h1}y_h + \beta_{h2}(y_h - x_h)] = \beta_{i1}y_i, \quad \boldsymbol{A}_j\boldsymbol{x}+b_j - \sum_{h=0}^{i-1}[\beta_{h1}y_h + \beta_{h2}(y_h - x_h)] = \beta_{i2}(y_i - x_i),$$

where we assume $\beta_{01}y_0 + \beta_{02}(y_0 - x_0) = 0$ for notational convenience. Generally, we have the value of β_{i1} and β_{i2} as follows,

$$\beta_{i1} = \min\left\{\frac{\boldsymbol{A}_j\boldsymbol{x}' + b_j - \sum_{h=0}^{i-1}[\beta_{h1}y'_h + \beta_{h2}(y'_h - x'_h)]}{y'_i}\right\}, \quad \beta_{i2} = \min\left\{\frac{\boldsymbol{A}_j\boldsymbol{x}' + b_j - \sum_{h=0}^{i-1}[\beta_{h1}y'_h + \beta_{h2}(y'_h - x'_h)]}{y'_i - x'_i}\right\}. \tag{10}$$

They are calculated in the i-th iteration in Algo 1.

5.3 Correctness of Innovation #2

Recall that due to the expensiveness of the conversion from the H-representation of a polytope to its V-representation (i.e., $GetVertices$ in Algo. 1) for getting its vertices (also discussed in Sec. 6.2.2), Innovation #2 proposes an incremental manner to determine the values of β_{ik}s by vertices. Theorem 5.5 guarantees the soundness of WraLU with Innovation #2.

Theorem 5.5 (β_{ik}s Identification through Vertices). *Using Formula (10), β_{ik}s $k \in [2]$ can be identified by the vertices transformed from vertices of X to derive a sound constraint.*

Proof. Due to symmetricity, we only discuss the case of determining the value of β_{i1}, and β_{i2} has a similar proof. We need to prove that only the vertices of (1) in Theorem 3.4 is necessary. The origin of (3) cannot determine the value of β_{i1} as the origin makes the denominator zero. Therefore, we only need to discuss the cases of the type (2).

First, the vertices with $y_i = 0$ cannot determine β_{i1}.

Next, if a vertex v satisfies $y_i \neq 0$ and it is not of type (1), we will prove that another vertex v' of type (1) determines the same or a smaller value of β_{i1}.

If β_{i1} is zero, then the numerator of Formula (10) is zero. There exists a vertex v' also on the hyperplane specified by the numerator, reaching the same value of zero.

If β_{i1} is not zero, then consider that the projection of v is on the intersection of one axis and an $(n-1)$-face (edge in high-dimensional space) of X by Theorem. 3.4. Because v is on one axis, it makes $x_h = y_h = 0$ for one dimension h $(1 \leq h \leq i-1)$. As there is another vertex v' with $x_h = 0$ on the same $(n-1)$-face as v and v' does not necessarily have $y_h = 0$ but $y_h \leq 0$, v' makes β take a value that is not greater than that determined by v due to the non-negative coefficient of y_h (Theorem 5.4). This is a contradiction to that v gets the smallest β_{i1}.

Hence, β_{ik}s can be identified by the vertices transformed from vertices of X. □

5.4 Optimality

Incompleteness. We remark that the approximation of WraLU leads to an over-approximation of the exact ReLU hull. While we give experimental evaluation in Sec. 6.2.2, in this section we present such an example and use it to analyze the root cause that WraLU cannot guarantee completeness.

Consider the input polytope X defined as a triangle $X = \{1 + x_1 \geq 0,\ 1 + x_2 \geq 0,\ 2 - x_1 - x_2 \geq 0\}$. Upon processing the y_1 coordinate, we obtain $\widetilde{M}_1 = \{2 - x_1 - x_2 \geq 0,\ 1 + x_1 - \frac{4}{3}y_1 \geq 0,\ 1 + x_2 \geq 0,\ y_1 \geq 0,\ -x_1 + y_1 \geq 0\}$. Next, the convex approximation $\widetilde{M}_{1,2}$ is calculated as $\widetilde{M}_{1,2} = \{2 - x_1 - x_2 \geq 0,\ 1 + x_1 - \frac{4}{3}y_1 \geq 0,\ 1 + x_2 - \frac{4}{3}y_2 \geq 0,\ y_1 \geq 0,\ y_2 \geq 0,\ -x_1 + y_1 \geq 0,\ -x_2 + y_2 \geq 0\}$. Compared to $\widetilde{M}_{1,2}$, the ReLU hull contains two additional constraints, $2 - x_1 + 2x_2 + y_1 - 3y_2 \geq 0$ and $2 + 2x_1 - x_2 - 3y_1 + y_2 \geq 0$.

To explain the reason, we consider the constraint $2 - x_1 + 2x_2 + y_1 - 3y_2 \geq 0$ and it is specified by one upper face. This constraint is equal to another form of $2 - x_1 - x_2 \geq -y_1 + 3(y_2 - x_2)$. However, $2 - x_1 - x_2 \geq -y_1$ is not a constraint of $\widetilde{M}_1$. Therefore, this constraint cannot be obtained by WraLU when using $\widetilde{M}_1$ to extend to coordinate y_2. Moreover, if we calculate coordinate y_2 firstly, the same result $\widetilde{M}_2 = \widetilde{M}_1$ still does not contain this constraint.

To obtain these constraints that have not been included, we take the first one as an example and consider another form of $2 - x_1 - x_2 \geq -y_1 + 3(y_2 - x_2)$. It is consistent with the general form in Theorem 3.5, but with a negative coefficient of y_1. Note that WraLU identifies non-negative coefficients (proved in Theorem 5.4). In fact, this constraint is not determined by one 3d face and a point, but one 2d face, $\{2 - x_1 - x_2 = 0,\ y_1 = 0,\ y_2 - x_2 = 0\} \cap M_1$, and two vertices, $(0, 2, 0, 2)$ introduced by X, and $(-1, 3, 0, 3)$ transformed from X. We only consider a special type of constraint that is determined by one $(n + i - 2)$-face and one vertex transformed from vertices of X, and due to this, WraLU is an approximate method.

Tightness. Despite its incompleteness, WraLU can provide a tight approximation. First, WraLU keeps the constraints derived by the exact lower faces. Next, the constructed upper-face constraints are determined by the edges and vertices of M, i.e., they contain these exact edges and vertices. Furthermore, all upper-face constraints have non-negative coefficients of y_is, so each of them provides an upper bound to each y_i by Formula (4). WraLU obtains the exact convex hulls of dimensions up to three, same as in SBLM+PDDM [Müller et al. 2022], because it constructs new faces only by pairing a vertex and one edge (n-face in high dimension).

6 EVALUATION

We have developed WraLU as a tool, and evaluated its performance on ReLU hull approximation (i.e., an intrinsic study) and ReLU-based neural network verification (i.e., an extrinsic study). In this section, we detail our implementation, experiments and results.

6.1 Implementation

WraLU is implemented in Python. It imports *pycddlib* [pyc 2023; Fukuda 2003] to convert the H-representation of a polytope to its V-representation (i.e., *GetVertices* in line 1 of Algo. 1) by the double description algorithm [Fukuda and Prodon 1995], and *Gurobi* [Gurobi Optimization, LLC 2023] to solve linear programming problems.

We also integrated WraLU into PRIMA, the state-of-the-art multi-neuron NN verifier based on the ReLU hull approximation, to explore its capability in verifying ReLU NNs. Considering that the output coordinate order may affect the precision of our method, we have instantiated two special orders, a by-default order which is the same as the PRIMA and the other being its reverse. In the remaining of this section, we use WraLU, WraLUX to denote our tool equipped with the default order and both orders, respectively, and we further use PRIMA+WraLU, PRIMA+WraLUX to denote our integration with PRIMA. All reported experiments are conducted on a workstation equipped with one AMD EPYC 7702P 64-core 2.00GHz CPU with 100G of main memory.

6.2 An Intrinsic Study: ReLU Hull Approximation

We first evaluate the performance of both WraLU and WraLUX, on four metrics including precision, efficiency, constraints complexity, and scalability.

6.2.1 Experimental Settings. Our experimental settings are summarized as follows.
Baseline. We compare WraLU and WraLUX with three methods, including two approximate methods, (1) the triangle relaxation method used in [Ehlers 2017] and (2) the SBLM+PDDM method used in PRIMA [Müller et al. 2022], and (3) one exact method used in [Singh et al. 2019a].
Evaluation metrics. The input for all methods above is a bounded polytope in H-representation, and the output is a convex polytope in H-representation over-approximating the ReLU hull. The performance is broken down into the following metrics, (1) *precision* by the volume of the resulting polytope, (2) *efficiency* by the calculation time, (3) *constraints complexity* by the number of generated constraints regarding the approximation, and (4) *scalability* by the capability to handle high dimensions.
Input polytope generation. We first prepare input polytopes for these methods. An input polytope sample is randomly generated through constructing its constraints. We employ an n-dimensional input polytope with $3^n/4^n$ random constraints ($3^n - 1$ constraints in the current framework of PRIMA to construct multi-neuron constraints) for generating the samples. A constraint $\boldsymbol{ax} + b \geq 0$ is determined by sampling coefficients, i.e. $\boldsymbol{a}$ and b. We sample each element of the vector $\boldsymbol{a}$ from a uniform distribution over $[-1, 1]$, and the scalar b from a uniform distribution over $(0, 1]$, ensuring that the input polytope contains the origin as an inner point without leading to trivial

Table 1. Performance of exact/approximate algorithms for ReLU hull

Input Dim.	Exact Method		Triangle Relax.		SBLM+PDDM		WraLU		WraLUX	
	Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.
Relative Volume of Resulting Polytope (3^n input constraints)										
2	0.002668	0.001668	0.009028	0.004622	0.006340	0.006379	**0.003515**	0.002332	**0.003129**	0.001920
3	0.000074	0.000047	0.001131	0.000410	0.000306	0.000178	**0.000152**	0.000099	**0.000113**	0.000066
4	0.000002	0.000001	0.000152	0.000067	0.000012	0.000007	**0.000008**	0.000006	**0.000005**	0.000004
Relative Volume of Resulting Polytope (4^n input constraints)										
2	0.002162	0.001620	0.008155	0.004872	0.005141	0.004360	**0.002828**	0.001905	**0.002620**	0.001747
3	0.000075	0.000035	0.001139	0.000355	0.000279	0.000121	**0.000156**	0.000078	**0.000118**	0.000058
4	0.000001	0.000001	0.000149	0.000082	0.000010	0.000007	**0.000007**	0.000005	**0.000003**	0.000002
Runtime(s) (3^n input constraints)										
2	0.001390	0.000225	0.000035	0.000004	**0.000158**	0.000012	0.000208	0.000012	0.000320	0.000012
3	0.381382	0.212652	0.000035	0.000004	0.001095	0.000081	**0.000364**	0.000032	**0.000531**	0.000036
4	596.165636	458.522783	0.000033	0.000005	0.020614	0.003001	**0.000762**	0.000130	**0.001008**	0.000142
Runtime(*s*) (4^n input constraints)										
2	0.001658	0.000377	0.000036	0.000009	0.000167	0.000032	**0.000348**	0.001635	**0.000534**	0.000032
3	0.314481	0.155117	0.000033	0.000005	0.001148	0.000069	**0.000548**	0.000043	**0.000731**	0.000045
4	627.042926	408.959439	0.000034	0.000006	0.024954	0.003071	**0.001996**	0.000301	**0.002354**	0.000329
Output Constraints Number (3^n input constraints)										
2	12.50	2.13	6.00	0.00	**12.30**	1.04	16.00	0.00	28.00	0.00
3	393.50	145.33	9.00	0.00	52.80	6.29	**39.00**	0.00	72.00	0.00
4	38034.50	19770.21	12.00	0.00	281.80	48.25	**80.00**	0.00	**152.00**	0.00
Output Constraints Number (4^n input constraints)										
2	13.20	2.91	6.00	0.00	**12.67**	0.94	24.00	0.00	44.00	0.00
3	347.30	126.30	9.00	0.00	**52.27**	4.34	93.00	0.00	180.00	0.00
4	38717.27	16289.49	12.00	0.00	301.37	54.31	**272.00**	0.00	536.00	0.00

cases. An additional interval boundary $[-5, 5]$, which aligns with most cases in real-world NNs, is set for all coordinates to ensure that the input polytope is bounded (see Definition 2.2). Note that SBLM+PDDM only supports the high-dimensional octahedrons as inputs [Müller et al. 2022]. Therefore, a prepossessing to obtain an octahedral over-approximation is needed for SBLM+PDDM (not counted into the runtime measurement).

6.2.2 Experimental Results and Analysis. Table 1 lists the performance of all methods on three metrics, i.e., precision, efficiency and constraints complexity, under an input dimension of 2–4. We present the mean and standard deviation for each metric, where the mean value indicates the overall performance and the standard deviation measures the variability of results. A smaller standard deviation means a stable performance on random samples. The scalability of our methods is presented in Fig. 4. It is measured by the time consumption in handling high-dimensional polytopes up to an input dimension of 8.

Precision. Given that all these approximate methods are to over-approximate the exact ReLU hull, the generated approximations by all methods have a greater volume than that of the hull generated by the exact method. A tight over-approximation has a less volume and is more closed to that of the exact ReLU hull. Since computing the exact volume analytically is hard, we use random sampling to estimate the volumes of resulting polytopes. Specifically, random points are sampled in a box region bounded by the bounds of variables of input polytope (y_i take the bounds of x_i), and we estimate the volume of polyhedron $\widetilde{M}$ by volume$(\widetilde{M}) = \frac{\#(\text{points in } \widetilde{M})}{\#(\text{all the sampled points})}$.

As shown in Table 1, among all approximate methods, our two methods demonstrate significant advantages on both mean volumes and standard deviation, indicating a higher level of precision than triangle relaxation (0.05X–0.2X) and SBLM+PDDM (0.4X–0.7X), and a stable performance. WraLUX has lower mean volumes than WraLU by up to 0.5X improvements on precision, and a less standard deviation than WraLU too. This enhancement is because WraLUX's two orders with an additional reverse order generate two over-approximations, and intersecting them yields a tighter approximation.

Efficiency. As shown in Table 1, our methods significantly outperform the exact method, which takes the longest time, by completing the task 10X–10^6X faster. In most cases, they outperform SBLM+PDDM, especially for the higher dimensions, by up to 30X faster. The triangle relaxation achieves the shortest time. This can be attributed to its single-neuron approximate method that does not consider the dependency between multiple neurons. Its trade-off may be weaker though, compared with SBLM+PDDM and our methods, as it has innegligible sacrifices on the precision.

Constraints Complexity. The constraint number of the resulting polytopes can be an indicator of the simplicity of polytopes. A polytope with fewer constraints can be beneficial when using linear programming for the subsequent verification. The experimental results in Table 1 show that the exact method has the most constraints and that of triangle relaxation is the least. The triangle relaxation has a fixed constraint number of $3n$, where n is the input dimension. The constraint number of our methods is closely dependent on the input polytopes. In particular, WraLU has the same constraints number with the input polytopes plus constraints of $y_i \geq x_i$ and $y_i \geq 0$, and WraLUX has a double number because it uses two variable orders. Therefore, WraLU has $m + 2n$ constraints and WraLUX has $2m + 2n$, where m is the constraints number of the input polytope and $2n$ is the number of lower faces. Our methods reduce up to 99% for the exact method and up to 30% for SBLM+PDDM.

Scalability. It is important for an approximation method to scale up on high-dimensional inputs, considering that real-world problems such as NN verification typically handle high-dimensional data. As shown in Fig. 4, WraLU and WraLUX are capable of resolving higher dimensions, by completing the approximation task within 10s for 8d scenarios. We restrict the input dimension to a maximum of 8, as in typical application domains, e.g., NNs, a large number of constraints may contain too many redundant constraints under most generalization settings. In contrast, the exact method takes 10min to resolve a polytope of an input dimension of 4 (not shown in Fig. 4 due to the significant difference in scale), which may not be acceptable in solving real-world problems like NN verification. The current implementation of SBLM+PDDM does not support the input dimension higher than 4 either. As the approximation of the triangle relaxation is too coarse, its scalability is not measured here.

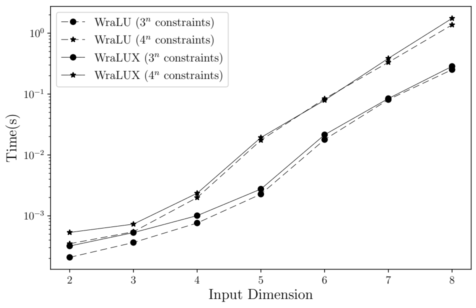


Fig. 4. Average total runtime of WraLU and WraLUX with different input dimensions

6.3 An Extrinsic Study: ReLU Neural Network Verification

This section evaluates the performance of our methods in verifying NNs, in particular, generating multi-neuron constraints as the over-approximation of the ReLU hulls of NNs. We integrate WraLU and WraLUX into PRIMA [Müller et al. 2022], the state-of-the-art multi-neuron NN verifier, denoted by PRIMA+WraLU and PRIMA+WraLUX, respectively. We first evaluate their overall performance on verifying NNs, and then analyze the contribution of our methods to the efficiency of over-approximating ReLU hull and solving linear programming problems.

Table 2. ReLU neural networks architectures used in evaluation

MNIST Network (ERAN)	#Neurons	#Hidden Layers	CIFAR10 Network (ERAN)	#Neurons	#Hidden Layers
FCTiny	110	2	FCTiny	410	4
FCSmall	510	5	FCSmall	610	4
FCBig	1610	8	FCBig	1810	9
ConvSmall	3604	3	ConvSmall	4852	3
ConvBig[a]	48064	6	ConvBig[a]	62464	6
MNIST Network (Ours)	**#Neurons**	**#Hidden Layers**	**CIFAR10 Network (Ours)**	**#Neurons**	**#Hidden Layers**
FCWide1	2058	2	FCWide1	2058	2
FCWide2	4106	4	FCWide2	4106	4
ConvBig2	48064	6	ConvBig2	62464	6
FMNIST Network (Ours)	**#Neurons**	**#Hidden Layers**	**EMNIST Network (Ours)**	**#Neurons**	**#Hidden Layers**
FCWide1	2058	2	FCWide1	2058	2
FCWide2	4106	4	FCWide2	4106	4
ConvBig2[a]	48064	6	ConvBig2[a]	48080	6

[a] Trained by DiffAI [Mirman et al. 2018].

6.3.1 Experimental Settings. In this section, we outline the baseline, evaluation metrics, and benchmarks. Following PRIMA, we adopt the neuron grouping strategy for approximating a high-dimensional ReLU hull, and also utilize DeepPoly (the CPU version) to obtain input polytopes [Müller et al. 2022]. We also tackle numerical issues that occur when a massive number of floating-point numbers are involved in the computation. All experiments are conducted on the same device as our intrinsic study.

Baseline. We compare PRIMA+WraLU and PRIMA+WraLUX with both types of approximate methods. For single-neuron methods, we select four bound propagation approaches, including DeepPoly [Singh et al. 2019b], DeepZ [Singh et al. 2018], CROWN and α-CROWN (with default settings) [Zhang et al. 2018], and triangle relaxation [Ehlers 2017]. For multi-neuron methods, we compare our methods with PRIMA [Müller et al. 2022]. We note that another multi-neuron verifier OptC2V [Tjandraatmadja et al. 2020] is not compared with, as PRIMA paper reports that PRIMA outperforms it.

Evaluation metrics. We evaluate the performance of all these methods on verifying local robustness with a specified l_∞ perturbation. An ϵ of perturbation radius is specified for each of the networks. The number of verified samples in the first 100 correctly classified samples and total runtime are two metrics to measure the overall performance. To further evaluate the contribution to the efficiency of our methods, we also measure the constraint number, and the total runtime of calculating ReLU hulls and solving linear programming problems.

Benchmarks. We conduct experiments on diverse network architectures, as outlined in Table 2. First, 10 representative benchmarks are taken from the ERAN project [era 2022], including 6 fully-connected and 4 convolutional ReLU NNs, ranging from 110 to 62,464 neurons. They are trained on the MNIST [Deng 2012] or CIFAR10 [Krizhevsky et al. 2009]. Second, to present the performance for large-scale network architectures, 8 fully-connected and 4 convolutional networks trained on MNIST [Deng 2012], CIFAR10 [Krizhevsky et al. 2009], FMNIST (Fashion-MNIST) [Xiao et al. 2017] and EMNIST [Cohen et al. 2017] (classification task of recognizing 26 letters) are used.

Neuron Grouping Strategy. While WraLU and WraLUX have stronger scalability in handling high dimensions, the exponential complexity, and numerous constraints make the computation impractical for all neurons in a layer. Therefore, we adopt the neuron grouping strategy introduced by PRIMA [Müller et al. 2022]. It combines neurons within the same layer into smaller groups, forming a cover for the layer. The constraints of all groups are then combined to approximate the entire layer. Three hyperparameters, namely n_s, k, and s, are utilized to control the grouping [Müller et al. 2022; Singh et al. 2019a], where n_s denotes the size of each partition, k determines the input

Table 3. Verifiable samples number and total runtime on MNIST-FCTiny under different grouping settings

n_s	k	s	**#Group**	**PRIMA**		**PRIMA+WraLU**		**PRIMA+WraLUX**	
				#Veri.	**Time(s)**	**#Veri.**	**Time(s)**	**#Veri.**	**Time(s)**
-	1[a]	-	-	44	20.67	-	-	-	-
20	3	1	6770	56	111.85	**58**	**70.19**	**58**	**69.45**
20	3	2	149527	62	823.52	61	**609.53**	62	**838.62**
20	4	1	3334	58	181.51	**60**	**90.29**	**61**	**123.79**
20	4	2	25678	63	769.24	**64**	**460.17**	**65**	**650.80**
20	4	3	600149	56	13312.75	54	**7568.22**	48	**8724.09**
20	5	1	1652	-	-	56	392.38	57	410.99
20	5	2	9152	-	-	66	1938.68	68	2077.36
20	5	3	69255	-	-	67	14754.99	66	15209.94
100	3	1	12987	56	183.66	**58**	**103.08**	**58**	**124.98**
100	3	2	461375	64	3651.10	**65**	**2058.82**	**65**	**2620.46**
100	4	1	5500	60	258.15	**62**	**145.26**	**63**	**169.91**
100	4	2	92298	65	3640.42	**66**	**1790.98**	**66**	**1908.12**
100	5	1	2768	-	-	58	628.18	60	694.55
100	5	2	31415	-	-	71	7067.30	66	7675.84
100	5	3	494843	-	-	49	115974.83	48	119965.97

[a] This setting is equivalent to triangle relaxation.

dimension of each neuron group within each partition, and s represents the maximum overlapping size between any two groups.

We use MNIST-FCTiny under different grouping settings to determine suitable hyperparameters for our benchmarking. As shown in Table 3, increasing the number of groups leads to more verified samples (5–10) but sacrifices efficiency. Our methods support higher dimensions and have the best of 71 verified samples than 65 of PRIMA, and also have a significant advantage in runtime (1.5X–2X faster than PRIMA). The verification with the setting of $(n_s = 20, k = 3, s = 1)$ has the shortest runtime, and $(n_s = 100, k = 4, s = 1)$ leads to more verified samples within a relatively shorter runtime compared to other settings. Therefore, we select two typical scenarios, $(n_s = 20, k = 3)$ and $(n_s = 100, k = 4)$, both with $s = 1$, for the following evaluation.

Mitigation of Numerical Issues. The validity of linear programming-based verification is sometimes compromised due to infeasible regions resulting from numerical issues in handling floating-point numbers, which may mis-indicate the absence of solutions. They occur often when dealing with large networks and numerous neuron groups that involve hundreds of thousands of constraints. To mitigate them, we enforce the following settings for all methods. On the ReLU hull approximation:

1) When the exceptions reported by *pycddlib* are caught, WraLU redoes the computation in fraction number type.

2) Because the input polytope has defined constraints, the number of vertices should be stable. If WraLU detects abnormal numbers of vertices, it redoes the computation in fraction number type.

3) We utilize PRIMA's hyperparameter named *cutoff*, to exclude neurons with too small bounds from ReLU hull calculations. Specifically, if the bounds (lower bound l and upper bound u) of a neuron satisfy $|lu| <$ cutoff, then this neuron is excluded.

On the linear programming problem solving, we employ a more stringent numerical focus in *Gurobi* to reduce the occurrence of possible numerical issues.

6.3.2 *Performance on ERAN Benchmarks.*

This section evaluates the performance of all methods on the ERAN benchmarks.

Overall Performance. Table 4 presents the performance of those approaches using single-neuron constraints. DeepPoly achieves an obviously superior performance on the number of verified samples compared to other approaches. Therefore, we further choose DeepPoly to provide the input polytopes in H-representation for calculating ReLU hull approximations.

Table 5 compares the performance of our methods and PRIMA. Between the two different grouping strategies, the precision under $(n_s = 100, k = 4)$ is mostly better than $(n_s = 20, k = 3)$

Table 4. Verifiable samples number and total runtime of different methods using single-neuron constraints (ERAN benchmarks)

Dataset	**Network**	ϵ	**CROWN**[a]		**α-CROWN**		**DeepZ**		**DeepPoly**	
			#Veri.	**Time**(s)	**#Veri.**	**Time**(s)	**#Veri.**	**Time**(s)	**#Veri.**	**Time**(s)
MNIST	FCTiny	0.03	3	3.64	4	70.4	38	35.6	44	4.91
	FCSmall	0.019	4	7.04	6	243.39	35	70.03	53	18.97
	FCBig	0.012	8	11.94	9	709.24	29	351.97	53	85.07
	ConvSmall	0.1	8	7.18	12	936.68	35	52.41	45	39.26
	ConvBig	0.305	14	496.79	-[b]	-[b]	8	11386.04	49	1797.22
CIFAR10	FCTiny	0.001	45	6.71	46	396.97	41	336.12	45	50.88
	FCSmall	0.0007	50	10.53	62	619.46	45	418.05	51	91.56
	FCBig	0.0008	53	18.19	56	1769.27	37	1469.24	53	393.57
	ConvSmall	0.004	49	9.27	51	1721.29	36	197.27	49	94.38
	ConvBig	0.007	47	563.38	-[b]	-[b]	45	10545.14	48	4430.09

[a] The discrepancy in the results of CROWN and DeepPoly is mainly due to differences in their by-default settings. DeepPoly takes the last ReLU layer into consideration and also clips the input perturbed intervals to $[0, 1]$, leading to tight input intervals in some cases.

[b] A single sample timeout lasts for one hour.

Table 5. Verifiable samples number and total runtime of different methods using multi-neuron constraints (ERAN benchmarks)

Dataset	**Network**	ϵ	**PRIMA**		**PRIMA+WraLU**		**PRIMA+WraLUX**	
			$(n_s = 20, k = 3)$	$(n_s = 100, k = 4)$	$(n_s = 20, k = 3)$	$(n_s = 100, k = 4)$	$(n_s = 20, k = 3)$	$(n_s = 100, k = 4)$
MNIST	FCTiny	0.03	44+12(111.85s)[a]	44+16(258.15s)	44+**14**(**70.19**s)	44+**18**(**145.26**s)	44+**14**(**69.45**s)	44+**19**(**169.91**s)
	FCSmall	0.019	53+7(510.56s)	53+8(2649.36s)	53+7(**465.81**s)	53+**10**(**2149.3**s)	53+**10**(534.80s)	53+**10**(2830.25s)
	FCBig	0.012	53+5(1799.22s)	53+6(13587.90s)	53+5(**1337.65**s)	53+6(**11086.70**s)	53+5(**1711.46**s)	53+6(**12552.22**s)
	ConvSmall	0.1	45+18(388.52s)	45+20(1072.8s)	45+18(**375.34**s)	45+**21**(**1055.21**s)	45+**19**(429.95s)	45+**23**(1491.31s)
	ConvBig[b]	0.305	49+**6**(4254.30s)	46+**7**(4380.95s)	46+5(**3575.15**s)	45+1(**3779.91**s)	49+4(**3274.40**s)	49+0(**3200.66**s)
CIFAR10	FCTiny	0.001	45+4(411.50s)	45+4(528.23s)	45+4(**305.33**s)	45+4(**402.28**s)	45+**5**(**318.04**s)	45+**5**(**423.02**s)
	FCSmall	0.0007	51+14(579.29s)	51+14(791.62s)	51+13(**433.64**s)	51+13(**603.78**s)	51+14(**461.21**s)	51+14(**657.24**s)
	FCBig	0.0008	53+**5**(3531.99s)	53+6(15404.19s)	53+4(**3183.01**s)	53+**9**(**13176.7**s)	53+4(**3388.72**s)	53+**9**(**13920.71**s)
	ConvSmall	0.004	49+8(**821.59**s)	49+10(**2131.12**s)	49+8(856.66s)	49+10(2220.91s)	49+8(947.48s)	49+**11**(2673.93s)
	ConvBig	0.007	48+3(11029.75s)	48+3(11724.91s)	48+3(**9865.48**s)	48+3(**9738.36**s)	48+3(**7990.02**s)	48+3(**8279.24**s)

[a] $m + n$ stands for n more networks are verified besides m verified by DeepPoly. Numbers in brackets refer to the total runtime. Numbers in bold refer to those cases where our methods outperform PRIMA. The ϵs used are the same in Table 4.

[b] Results include those affected by numerical issues (as unsuccessfully verified samples).

with more up to 4 verified samples for PRIMA and up to 5 for our methods. Between fully-connected and convolutional networks, our methods has an advantage on fully-connected networks, because the computation of ReLU hulls takes a smaller proportion for the structures of convolutional networks. PRIMA+WraLUX has a slightly better precision than PRIMA+WraLU (up to 2 more samples) but sacrifices efficiency. Overall, our methods achieve a competitive number of verified samples (up to 4 more) and shorter runtime (up to 1.2X faster), demonstrating higher precision and efficiency in most cases.

Performance on ReLU hull approximation and linear programming problem solving. We further analyze the performance of two key components of the verification, i.e., ReLU hull approximation and linear programming problem solving, to reveal the contribution to efficiency from WraLU and WraLUX. As shown in Table 6, our methods have fewer constraints (up to 50%) and less computation time (up to 50%) on both ReLU hull approximation and linear programming solving in most cases. We highlight that the constraint amount reduction by WraLU indeed leads to the runtime reduction in linear programming solving.

6.3.3 Performance on Complex Benchmarks. To further explore the efficiency contribution of WraLU and WraLUX, we conduct performance comparisons with PRIMA, using the benchmarks that are large-scale networks with more datasets. The purpose of using complex benchmarks

Table 6. Constraints number and total runtime of ReLU hull approximation and linear programming problem solving (ERAN Benchmarks)

ReLU Hull Approximation							
Dataset	**Network**	**PRIMA**		**PRIMA+WraLU**		**PRIMA+WraLUX**	
		$(n_s = 20, k = 3)$	$(n_s = 100, k = 4)$	$(n_s = 20, k = 3)$	$(n_s = 100, k = 4)$	$(n_s = 20, k = 3)$	$(n_s = 100, k = 4)$
MNIST	FCTiny	322726(49.96s)	1073743(113.74s)[a]	**175783**(**30.65**s)	**439926**(**61.09**s)	351566(**31.30**s)	**879852**(**73.08**s)
	FCSmall	1454266(179.63s)	12753175(473.79s)	**746894**(**160.95**s)	**4631040**(**305.76**s)	1493788(**174.22**s)	**9262080**(**334.72**s)
	FCBig	3617811(321.93s)	40312622(1435.33s)	**1878972**(**179.06**s)	**15366876**(**674.98**s)	3757944(**229.24**s)	**30733752**(**792.06**s)
	ConvSmall	556314(95.72s)	3168860(132.83s)	**339590**(**87.61**s)	**1623120**(**93.62**s)	679180(101.80s)	3246240(**118.47**s)
	ConvBig	316714(172.70s)	**922528**(220.99s)	**296112**(**136.59**s)	1417310(**180.50**s)	592224(**137.36**s)	2834620(**185.41**s)
CIFAR10	FCTiny	224466(144.20s)	517579(177.29s)	**125130**(**109.92**s)	**222995**(**120.82**s)	250260(**113.26**s)	**445990**(**120.34**s)
	FCSmall	277527(213.01s)	708092(250.32s)	**153242**(**152.93**s)	**291466**(**180.16**s)	306484(**160.28**s)	**582932**(**176.35**s)
	FCBig	2056807(378.40s)	16985398(948.79s)	**1123394**(**278.46**s)	**6954116**(**466.67**s)	2246788(**295.91**s)	13908232(**525.06**s)
	ConvSmall	982907(115.04s)	5931802(260.55s)	**753304**(**104.28**s)	**5403332**(**153.34**s)	1506608(**102.94**s)	10806664(**171.06**s)
	ConvBig	65986(109.64s)	188261(116.60s)	**35238**(**83.64**s)	**66956**(**85.93**s)	**70476**(**84.4**s)	**133912**(**74.78**s)

Linear Programming Problem Solving							
Datasets	**Networks**	**PRIMA**		**PRIMA+WraLU**		**PRIMA+WraLUX**	
		$(n_s = 20, k = 3)$	$(n_s = 100, k = 4)$	$(n_s = 20, k = 3)$	$(n_s = 100, k = 4)$	$(n_s = 20, k = 3)$	$(n_s = 100, k = 4)$
MNIST	FCTiny	342180(19.92s)	1093197(58.54s)	**195237**(**10.16**s)	**459380**(**27.74**s)	371020(**12.38**s)	**899306**(**38.87**s)
	FCSmall	1533044(129.45s)	12831953(1029.76s)	**825672**(**119.04**s)	**4709818**(**804.63**s)	1572566(147.93s)	**9340858**(1443.05s)
	FCBig	3855021(324.92s)	40549832(3785.54s)	**2116182**(**262.24**s)	**15604086**(**3181.13**s)	3995154(412.32s)	**30970962**(4233.79s)
	ConvSmall	1013874(77.34s)	3626420(444.01s)	**797150**(**75.19**s)	**2080680**(496.18s)	1136740(106.79s)	3703800(914.47s)
	ConvBig	5247944(263.58s)	**5853758**(239.07s)	**5227342**(**167.17**s)	6348540(**237.34**s)	5523454(**148.38**s)	7765850(**191.01**s)
CIFAR10	FCTiny	275654(52.30s)	568767(73.64s)	**176318**(**28.09**s)	**274183**(**44.41**s)	301448(**34.68**s)	**497178**(**57.68**s)
	FCSmall	344953(83.02s)	775518(157.32s)	**220668**(**49.99**s)	**358892**(**95.81**s)	373910(**66.06**s)	**650358**(**140.07**s)
	FCBig	2268037(475.17s)	17196628(1928.25s)	**1334624**(501.78s)	**7165346**(**1529.20**s)	2458018(644.16s)	**14119462**(2181.44s)
	ConvSmall	1531973(180.001s)	6480868(509.16s)	**1302370**(302.11s)	**5952398**(867.70s)	2055674(380.63s)	11355730(1217.82s)
	ConvBig	6926252(1846.22s)	7048527(1791.34s)	**6895504**(1880.56s)	**6927222**(**1655.46**s)	6930742(**1400.40**s)	**6994178**(**1350.97**s)

[a] Numbers in brackets refer to the total runtime. Numbers in bold refer to those cases where our methods outperform PRIMA.

Table 7. Verifiable samples number and total runtime of different methods using multi-neuron constraints (complex benchmarks)

Dataset	**Network**	ϵ	**PRIMA** $(n_s = 20, k = 3)$	**PRIMA+WraLU** $(n_s = 20, k = 3)$	**PRIMA+WraLUX** $(n_s = 20, k = 3)$
MNIST	FCWide1	0.025	50+17(6188.12s)	50+**24**(**5607.3**s)	50+**24**(**6053.22**s)
	FCWide2	0.015	30+11(44870.75s)	30+**15**(**32644.09**s)	30+**16**(**37034.72**s)
	ConvBig2	0.025	43+**5**(119741.36s)	43+1(122674.86s)	43+0(**107418.68**s)
FMNIST	FCWide1	0.025	46+10(6290.07s)	46+**17**(**6096.72**s)	46+**14**(6293.40s)
	FCWide2	0.015	50+0(24298.26s)	50+0(**15744.50**s)	50+**3**(**18030.7**s)
	ConvBig2	0.007	49+10(68358.00s)	49+**11**(57990.18s)	49+10(**66434.00**s)
EMNIST	FCWide1	0.025	46+13(4340.34s)	46+**14**(**2273.37**s)	46+**15**(**3354.33**s)
	FCWide2	0.015	46+3(**7868.29**s)	46+**4**(7957.30s)	46+**4**(8502.64s)
	ConvBig2	0.014	52+13(59506.00s)	52+**17**(54365.94s)	52+0[a](59769.87s)
CIFAR10	FCWide1	0.002	45+11(6885.75s)	45+11(**4881.5**s)	45+11(**5231.68**s)
	FCWide2	0.0015	39+2(**9046.91**s)	39+2(9460.8s)	39+**3**(9653.2s)
	ConvBig2	0.003	47+2(107499.17s)	47+**6**(**81729.01**s)	47+0[a](96468.79s)

[a] Results include those affected by numerical issues (as unsuccessfully verified samples).

is to assess whether our methods have a stable superiority. Here we only present data for the $(ns = 20, k = 3)$ setting due to the frequent numerical issues in large-scale networks.

Overall Performance. As shown in Table 7, the results on complex benchmarks reveal a noticeable difference on the number of verified samples (up to 7), which is more significant than those on ERAN benchmarks. Besides, our methods demonstrate reduced total runtime (up to 50%) in the verification process. Even though more verified samples need to solve more linear programming problems, for more than half of the tested networks, our methods verify more samples in less runtime. For example, for MNIST-FCWide1 and FMNIST-FCWide1, there are 7 more verified samples in about 90% runtime of PRIMA.

Performance on ReLU hull approximation and linear programming problem solving. As shown in Table 8, our methods remain a stable superior in calculating a large number of ReLU hulls. Compared to PRIMA, our methods achieve a stable reduction (about 50%) in the number of

Table 8. Constraints number and total runtime of ReLU hull approximation and linear programming problem solving (complex benchmarks)

		ReLU Hull Approximation			Linear Programming Problem Solving		
Dataset	Network	PRIMA ($n_s = 20, k = 3$)	PRIMA+WraLU ($n_s = 20, k = 3$)	PRIMA+WraLUX ($n_s = 20, k = 3$)	PRIMA ($n_s = 20, k = 3$)	PRIMA+WraLU ($n_s = 20, k = 3$)	PRIMA+WraLUX ($n_s = 20, k = 3$)
MNIST	FCWide1	4895404(114.53s)	**2600032**(**56.72**s)	**5200064**(**74.7**s)	5221158(**1278.3**s)	**2925786**(1609.15s)	5525818(1828.55s)
	FCWide2	9720394(317.92s)	**5117022**(**131.45**s)	10234044(**219.9**s)	10652748(4343.14s)	**6049376**(**3497.6**s)	11166398(5797.76s)
	ConvBig2	14207024(320.27s)	**7849358**(**222.59**s)	15698716(**313.99**s)	20152072(6197.72s)	**13794406**(6320.54s)	21643764(**5746.41**s)
FMNIST	FCWide1	5234445(129.67s)	**2808744**(**80.85**s)	5617488(**84.66**s)	5569877(**1139.29**s)	**3144176**(1671.72s)	5952920(1801.62s)
	FCWide2	4726163(156.34s)	**2489652**(**77.35**s)	4979304(**123.71**s)	5314019(2539.68s)	**3077508**(**1836.36**s)	5567160(2681.64s)
	ConvBig2	4604229(207.14s)	**2620082**(**140.2**s)	5240164(215.44s)	9708645(6536.51s)	**7724498**(**6326.47**s)	10344580(7051.98s)
EMNIST	FCWide1	2006851(109.36s)	**1050260**(**66.94**s)	2100520(**57.89**s)	2298165(1060.67s)	**1341574**(**594.44**s)	2391834(**998.29**s)
	FCWide2	1218990(115.57s)	**641244**(**63.23**s)	1282488(**98.35**s)	1758148(1469.43s)	**1180402**(**1383.19**s)	1821646(1698.12s)
	ConvBig2	10587970(267.43s)	**5793356**(**172.94**s)	11566712(**251.3**s)	15679546(7011.14s)	**10874932**(9219.21s)	16658288(**4838.39**s)
CIFAR10	FCWide1	1104959(110.66s)	**577204**(**56.33**s)	1154408(**56.09**s)	1371373(2586.77s)	**843618**(**1743.33**s)	1420822(**2004.83**s)
	FCWide2	625033(71.20s)	**322786**(**48.60**s)	645572(72.37s)	1200931(1468.96s)	**898684**(**1256.22**s)	1221470(1490.39s)
	ConvBig2	5121883(254.68s)	**3028896**(**169.7**s)	6057792(261.03s)	12246587(**5712.87**s)	**10153600**(6643.28s)	13182496(**5338.12**s)

constraints, resulting in significant runtime reduction (30%–60%). These enhancements lead to an average time reduction of around 10%–20% when solving linear programming problems.

7 DISCUSSION ON GENERALIZATION OF WRALU

WraLU primarily focuses on ReLU due to its pivotal role in modern DNNs. Its key insight is to formulate a *convex polyhedron* as the tight over-approximation of the ReLU hull by reusing the linear pieces of the ReLU function as the *lower faces* and constructing *upper faces* that are adjacent to the lower faces. This insight stems from the piece-wise linearity and convexity of the ReLU function, which allow WraLU to treat its linear pieces as lower faces. Leveraging the same fundamental idea, WraLU can be extended to handle the following types of activation functions.

Piece-wise linear functions. WraLU can readily accommodate piece-wise linear convex functions, e.g., Leaky ReLU [Maas et al. 2013] and MaxPool [LeCun et al. 1998], by reusing their linear pieces as lower faces and constructing new faces using the method presented in Section 4. Symmetrically, piece-wise linear concave functions can also be handled.

Piece-wise non-linear functions. For non-linear functions, e.g., ELU [Clevert et al. 2016], no linear piece can be reused as the faces of their convex hull in their non-linear pieces. To apply WraLU, a linear piece-wised function has to be constructed as the upper bound or the lower bound to the transformation of the functions, so that the upper or lower faces can be constructed by this linear piece-wised function. For example, in ELU with $x \in [l, u]$, the linear piece-wise function of $y = x$ when $\mathrm{ELU}(l) \leq x \leq u$ and $y = l$ when $l \leq x \leq \mathrm{ELU}(l)$ can serve as the lower bound. The two pieces can be taken as the lower faces due to the convexity of this function. Another linear piece-wise function of $y = \frac{\mathrm{ELU}(l)}{l}(x - l) + \mathrm{ELU}(l)$ (when $l \leq x < 0$) and $y = x$ (when $0 \leq x \leq u$) can be taken as the upper bound. It has a similar shape to ReLU, so WraLU's upper faces identification (Section 4) can be used to construct the upper faces as the upper faces of the ELU hull.

Non-piece-wise differentiable functions. The solution for ELU hull discussed above can be adapted for other non-piece-wise differentiable functions, e.g., Tanh and Sigmoid [Goodfellow et al. 2016]. The key is to construct piece-wise linear functions to wrap these functions, and then WraLU's methods can be employed to find upper faces and lower faces.

8 RELATED WORK

Our work is closely related to convex hull/approximation computation and deterministic NN verification for ReLU functions. In this section, we present a review of representative studies in these two fields and also highlight the differences between our work and existing approaches.

Table 9. Comparison of deterministic approaches for neural network verification

Approach	Complete	Sound	ReLU Hull		
			Multi-neuron	Exact	High-dimensional
CROWN/α-CROWN [Zhang et al. 2018]	○	●	○	○	○
β-CROWN [Wang et al. 2021]	●	●	○	○	○
DeepPoly [Singh et al. 2019b]	○	●	○	○	○
DeepZ [Singh et al. 2018]	○	●	○	○	○
k-relu [Singh et al. 2019a]	○	●	●	●	○
PRIMA [Müller et al. 2022]	○	●	●	○	○
MN-BaB [Ferrari et al. 2022]	●	●	●	○	○
Ours	○	●	●	○	●

8.1 Convex Hull/Approximation Algorithms

Conventional algorithms. The convex hull problem is a classic and well-studied problem in computational geometry, and it has been tackled by a number of studies [Avis and Fukuda 1991, 1992; Barber et al. 1993; Chand and Kapur 1970; Dantzig and Thapa 2003; Edelsbrunner 1987; Fukuda and Prodon 1995; Jarvis 1973; Joswig 2003; Motzkin et al. 1953], typically the quickhull algorithm [Barber et al. 1993], the gift wrapping algorithm [Chand and Kapur 1970; Jarvis 1973], and the double description algorithm [Fukuda and Prodon 1995]. Among them, the gift-wrapping algorithm has an idea similar to ours, which discovers additional adjacent facets by rotating a hyperplane along the boundary.

Algorithms in NN verification. A few exact or approximate convex hull algorithms have been specifically applied for verifying neural networks. The existing approaches to using convex hull algorithms to construct ReLU hull or its approximation involve decomposing the orthants by piece-wise linearity property of the ReLU function to obtain the vertices of each linear piece, and then applying the convex hull/approximation algorithm to obtain the whole convex hull/approximation [Müller et al. 2022; Singh et al. 2019a]. The double description algorithm is used as the core calculation method in k-relu [Singh et al. 2019a], while PRIMA [Müller et al. 2022] introduces an approximate algorithm based on the idea of ray-shooting and incrementally calculating output coordinates. OptC2V [Tjandraatmadja et al. 2020] utilizes submodularity and convex geometry to get the convex approximation among all inputs and one output at the layer scale by linear programming. The bound step in PRIMA [Müller et al. 2022] and WraLU both extend the polytope by one dimension at a time. WraLU (in its innovation #2) removes the computation of unnecessary vertices that are newly generated by the ReLU function when extending the dimension, while the bound step used in PRIMA (in its SBLM) calculates each linear piece in each quadrant and its PDDM is then used to over-approximate the convex hull of these linear pieces by one more dimension at a time.

8.2 Neural Networks Verification

Complete and incomplete verification. NN verification has become increasingly important in critical domains [Meng et al. 2022]. Many verifiers use precise methods for complete verification and they can give a definitive answer to whether a given property satisfies or not. They typically utilize satisfaction modulo theory (SMT) [Ehlers 2017; Huang et al. 2017] or mixed integer linear programming (MILP) [Anderson et al. 2019; Botoeva et al. 2020; Bunel et al. 2019, 2018; De Palma et al. 2021; Wang et al. 2021; Xu et al. 2021] and extended simplex method [Katz et al. 2017, 2019] to parse the piece-wise linearity of the ReLU function. Due to the NP-hardness of verifying ReLU neural networks, these methods are often computationally expensive and thus are limited to small-scale NNs [Ehlers 2017; Weng et al. 2018].

Another line of research turns to approximate methods which sacrifice completeness for scalability. They typically propose to relax those ReLU neurons by a linear over-approximation, such as interval [Gowal et al. 2019], linear programming [Ehlers 2017], zonotope [Singh et al. 2018], polytopes [Singh et al. 2019b; Weng et al. 2018; Zhang et al. 2018], duality [Dvijotham et al. 2018a,b; Wong and Kolter 2018; Wong et al. 2018]. Recently, the branch-and-bound (BaB) paradigm has been proposed to break the barrier between complete and incomplete approaches. It incorporates incomplete approaches to accomplish a complete verification by decomposing the original problem into sub-problems [Bunel et al. 2018; Ferrari et al. 2022; Wang et al. 2021]. Our work can be naturally compatible with BaB and accomplish complete verification. Table 9 gives a comparison between our approach and some representative approaches.

Over-approximation to ReLU function. Over-approximating ReLU functions is essential to both complete and incomplete approaches. For single-neuron approximation, the most efficient approaches are based on bound propagation, like Fast-Lin [Weng et al. 2018], CROWN [Zhang et al. 2018], DeepPoly [Singh et al. 2019b], DeepZ [Singh et al. 2018], using only two linear lower and upper bounds to achieve bound propagation, and they can be accelerated using graphics processing units. Triangle relaxation is the tightest linear approximation for ReLU function [Ehlers 2017] but needs solving linear programming problems. Also, some studies [Dathathri et al. 2020; Raghunathan et al. 2018] apply the semidefinite programming to give a convex approximation to ReLU in a quadratic function, but they have limitations in scalability and computation time.

Besides the single neuron convex approximation, k-relu [Singh et al. 2019a] gets the constraints involving inputs and outputs of multiple neurons (3–4) as a group, and this has been applied to other general activation functions recently [Müller et al. 2022]. OptC2V [Tjandraatmadja et al. 2020] is an approach to calculate the constraints among all inputs and one output of a layer. In our work, we provide a technique to advance the approximate methods by efficiently approximating the ReLU hulls.

9 CONCLUSION

In this work, we propose a novel approach named WraLU for fast and precise over-approximation of the ReLU hull. The key insight is to formulate a convex polyhedron that wraps the ReLU hull, utilizing the linear pieces of the ReLU function as the lower faces and constructing upper faces adjacent to them. By leveraging the uniqueness of the ReLU hull in sharing linear pieces with the original ReLU function, the method efficiently determines the forms of adjacent faces and vertices, resulting in a tight approximation. We have implemented WraLU and evaluated its performance in terms of precision, efficiency, constraint complexity, and scalability for approximating ReLU hulls. Experimental results in large-scale ReLU-based NN verification showcase its high efficiency without compromising precision. We envision that WraLU will inspire future work to enhance the scalability and reliability of verification techniques, to advance the verification of neural networks with non-linear activation functions.

DATA AVAILABILITY STATEMENT

The source code of WraLU is available at https://github.com/UQ-Trust-Lab/WraLU.

ACKNOWLEDGMENTS

We thank our anonymous reviewers for their constructive comments. This work is partially supported by The University of Queensland (NSRSG 4018264-617225) and Australian Research Council Discovery Projects (DP230101196, DP240103068).

REFERENCES

2022. ERAN: ETH Robustness Analyzer for Neural Networks. https://github.com/eth-sri/eran

2023. pycddlib. https://pypi.org/project/pycddlib/

Ross Anderson, Joey Huchette, Christian Tjandraatmadja, and Juan Pablo Vielma. 2019. Strong Mixed-Integer Programming Formulations for Trained Neural Networks.. In *IPCO (Lecture Notes in Computer Science, Vol. 11480)*. Springer, 27–42.

David Avis and Komei Fukuda. 1991. A basis enumeration algorithm for linear systems with geometric applications. *Applied Mathematics Letters* 4, 5 (1991), 39–42.

David Avis and Komei Fukuda. 1992. A Pivoting Algorithm for Convex Hulls and Vertex Enumeration of Arrangements and Polyhedra. *Discret. Comput. Geom.* 8 (1992), 295–313.

C Bradford Barber, David P Dobkin, and Hannu Huhdanpaa. 1993. *The quickhull algorithm for convex hull.* Technical Report. Technical Report GCG53, The Geometry Center, MN.

Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. 2020. Efficient Verification of ReLU-Based Neural Networks via Dependency Analysis.. In *AAAI*. AAAI Press, 3291–3299.

Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and M. Pawan Kumar. 2019. Branch and Bound for Piecewise Linear Neural Network Verification. *CoRR* abs/1909.06588 (2019).

Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and Pawan Kumar Mudigonda. 2018. A Unified View of Piecewise Linear Neural Network Verification.. In *NeurIPS*. 4795–4804.

Donald R. Chand and Sham S. Kapur. 1970. An Algorithm for Convex Polytopes. *J. ACM* 17, 1 (1970), 78–86.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2016. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). http://arxiv.org/abs/1511.07289 arXiv:1511.07289 [cs].

G Cohen, S Afshar, J Tapson, and A van Schaik. 2017. Emnist: anextension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373* (2017).

George Bernard Dantzig and Mukund N Thapa. 2003. *Linear programming: Theory and extensions.* Vol. 2. Springer.

Sumanth Dathathri, Krishnamurthy Dvijotham, Alexey Kurakin, Aditi Raghunathan, Jonathan Uesato, Rudy Bunel, Shreya Shankar, Jacob Steinhardt, Ian J. Goodfellow, Percy Liang, and Pushmeet Kohli. 2020. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. *CoRR* abs/2010.11645 (2020).

Alessandro De Palma, Harkirat S Behl, Rudy Bunel, Philip Torr, and M Pawan Kumar. 2021. Scaling the convex barrier with active sets. In *Proceedings of the ICLR 2021 Conference*. Open Review.

Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

Krishnamurthy Dvijotham, Sven Gowal, Robert Stanforth, Relja Arandjelovic, Brendan O'Donoghue, Jonathan Uesato, and Pushmeet Kohli. 2018a. Training verified learners with learned verifiers. *arXiv preprint arXiv:1805.10265* (2018).

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A. Mann, and Pushmeet Kohli. 2018b. A Dual Approach to Scalable Verification of Deep Networks.. In *UAI*. AUAI Press, 550–559.

Herbert Edelsbrunner. 1987. *Algorithms in Combinatorial Geometry*. EATCS Monographs on Theoretical Computer Science, Vol. 10. Springer. I–XV, 1–423 pages. http://dx.doi.org/10.1007/978-3-642-61568-9

Rüdiger Ehlers. 2017. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks.. In *ATVA (Lecture Notes in Computer Science, Vol. 10482)*. Springer, 269–286.

Claudio Ferrari, Mark Niklas Muller, Nikola Jovanovic, and Martin Vechev. 2022. Complete verification via multi-neuron relaxation guided branch-and-bound. *arXiv preprint arXiv:2205.00263* (2022).

Komei Fukuda. 2003. Cddlib reference manual. *Report version 093a, McGill University, Montréal, Quebec, Canada* (2003).

Komei Fukuda and Alain Prodon. 1995. Double Description Method Revisited.. In *Combinatorics and Computer Science (Lecture Notes in Computer Science, Vol. 1120)*. Springer, 91–111.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation.. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 3–18.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Arthur Mann, and Pushmeet Kohli. 2019. Scalable Verified Training for Provably Robust Image Classification.. In *ICCV*. IEEE, 4841–4850.

Gurobi Optimization, LLC. 2023. Gurobi Optimizer Reference Manual. https://www.gurobi.com

Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 2017. Safety Verification of Deep Neural Networks.. In *CAV (1) (Lecture Notes in Computer Science, Vol. 10426)*. Springer, 3–29.

Ray A. Jarvis. 1973. On the Identification of the Convex Hull of a Finite Set of Points in the Plane. *Inf. Process. Lett.* 2, 1 (1973), 18–21.

Michael Joswig. 2003. Beneath-and-Beyond Revisited. In *Algebra, Geometry, and Software Systems*. Springer, 1–21.

Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks.. In *CAV (1) (Lecture Notes in Computer Science, Vol. 10426)*. Springer, 97–117.

Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks.. In *CAV (1) (Lecture Notes in Computer Science, Vol. 11561)*. Springer, 443–452.

Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. https://ieeexplore.ieee.org/abstract/document/726791/ Publisher: Ieee.

Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, Vol. 30. Atlanta, GA, 3.

Peter McMullen. 1970. The maximum numbers of faces of a convex polytope. *Mathematika* 17, 2 (1970), 179–184.

Mark Huasong Meng, Guangdong Bai, Sin Gee Teo, Zhe Hou, Yan Xiao, Yun Lin, and Jin Song Dong. 2022. Adversarial robustness of deep neural networks: A survey from a formal verification perspective. *IEEE Transactions on Dependable and Secure Computing* (2022).

Matthew Mirman, Timon Gehr, and Martin T. Vechev. 2018. Differentiable Abstract Interpretation for Provably Robust Neural Networks.. In *ICML (Proceedings of Machine Learning Research, Vol. 80)*. PMLR, 3575–3583.

T.S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall. 1953. The Double Description Method. In *Contributions to the Theory of Games II*. Princeton University Press.

Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin T Vechev. 2022. PRIMA: general and precise neural network certification via scalable convex hull approximations. *Proc. ACM Program. Lang.* 6, POPL (2022).

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. 2018. Semidefinite relaxations for certifying robustness to adversarial examples.. In *NeurIPS*. 10900–10910.

Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. 2018. Reachability Analysis of Deep Neural Networks with Provable Guarantees.. In *IJCAI*. ijcai.org, 2651–2659.

Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin T. Vechev. 2019a. Beyond the Single Neuron Convex Barrier for Neural Network Certification.. In *NeurIPS*. 15072–15083.

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T. Vechev. 2018. Fast and Effective Robustness Certification.. In *NeurIPS*. 10825–10836.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. 2019b. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.* 3, POPL (2019), 41:1–41:30.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks.. In *ICLR (Poster)*.

Christian Tjandraatmadja, Ross Anderson, Joey Huchette, Will Ma, Krunal Kishor Patel, and Juan Pablo Vielma. 2020. The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. *Advances in Neural Information Processing Systems* 33 (2020), 21675–21686.

Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming.. In *ICLR (Poster)*. OpenReview.net.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. 2021. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in Neural Information Processing Systems* 34 (2021), 29909–29921.

Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane S. Boning, and Inderjit S. Dhillon. 2018. Towards Fast Computation of Certified Robustness for ReLU Networks.. In *ICML (Proceedings of Machine Learning Research, Vol. 80)*. PMLR, 5273–5282.

Eric Wong and J. Zico Kolter. 2018. Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope.. In *ICML (Proceedings of Machine Learning Research, Vol. 80)*. PMLR, 5283–5292.

Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J Zico Kolter. 2018. Scaling provable adversarial defenses. *Advances in Neural Information Processing Systems* 31 (2018).

Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).

Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. 2021. Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers. In *International Conference on Learning Representation (ICLR)*.

Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. 2022. General cutting planes for bound-propagation-based neural network verification. *arXiv preprint arXiv:2208.05740* (2022).

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. 2018. Efficient Neural Network Robustness Certification with General Activation Functions.. In *NeurIPS*. 4944–4953.