Support Triangle Machine

JIAYING LI, OmniVision Technologies, Singapore
CHUNXUE HAO, China CITIC Bank, China

Approximations play a pivotal role in verifying deep neural networks (DNNs). Existing approaches typically rely on either single-neuron approximations (simpler to design but less precise) or multi-neuron approximations (higher precision but significantly more complex to construct). Between them, a notable gap exists.

This work bridges the gap. The idea is to lift single-neuron approximations into multi-neuron approximations with precision gain. To this end, we formulate the approximation transition as a novel problem, named *Convex Approximation Lifting* (CAL), and propose a constructive approach, *Support Triangle Machine* (STM), to solving it. STM is grounded in two core insights: *(i) there exists a simple geometric structure, called the support triangle, along with an efficient triangle lifting technique that connects single-neuron approximations and multi-neuron approximations*; and *(ii) typical single-neuron approximations can be easily decomposed into multiple atomically liftable components*. Specifically, given a CAL instance, STM constructs a multi-neuron approximation by iteratively processing each output coordinate. For each coordinate, it decomposes the single-neuron approximation into several linear parts, lifts each of them using the triangle lifting technique, and then synthesize an intermediate approximation, which later servers as input for the next iteration.

We theoretically prove the correctness of STM and empirically evaluate its performance on a variety of CAL problems and DNN verification tasks. Experimental results demonstrate STM's broad applicability, improved precision, and sustained efficiency. Beyond DNN verification, STM has the potential to facilitate approximation construction process in more general tasks, and we expect it to catalyze further research in related fields.

CCS Concepts: • **Software and its engineering → Formal software verification**; • **Theory of computation → Computational geometry**; **Numeric approximation algorithms**.

Additional Key Words and Phrases: Convex Approximation, Approximation Lifting, Support Triangle Machine

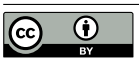
ACM Reference Format:
Jiaying Li and Chunxue Hao. 2025. Support Triangle Machine. *Proc. ACM Program. Lang.* 9, PLDI, Article 156 (June 2025), 24 pages. https://doi.org/10.1145/3729255

1 Introduction

Approximations are arguably one of the most powerful techniques for understanding complex systems and solving challenging problems in our world. Over the last decade, approximation technique has been adapted to verify deep neural networks (DNNs), a crucial problem towards addressing AI security risks, such as adversarial examples [36] and backdoor attacks [31, 38].

Verifying DNNs is inherently difficult, largely due to non-linear activation functions. In recent years, researchers have proposed a wide range of methods[13, 17, 23, 28, 29, 32–35, 37, 41], which can be broadly categorized as exact or approximate. Exact methods [23, 28, 37] are precise but limited in scalability, restricted to DNNs with piecewise linear activations like *ReLU*. Approximate methods [13, 17, 29, 32–35, 41], by contrast, are less precise but scalable, applicable to general activation functions such as *Sigmoid* and *Tanh*. Approximate methods can be further classified into two distinct groups: single-neuron approximation methods and multi-neuron approximation methods. The former

Authors' Contact Information: Jiaying Li, OmniVision Technologies, Singapore, Singapore, lijiaying1989@gmail.com; Chunxue Hao, China CITIC Bank, Beijing, China, haochunxue@citicbank.com.



ACM 2475-1421/2025/6-ART156
https://doi.org/10.1145/3729255

approximate each neuron independently, while the latter consider dependencies among multiple neurons (usually a group of neurons).

Motivation. Single-neuron approximation methods are efficient and relatively easy to design, but often suffer from limited precision. In contrast, multi-neuron approximation methods are more accurate but significantly more complicated to design and implement. This creates a pronounced gap between these two lines of approaches.

Our review of existing literature indicates limited efforts to bridge this gap. An exception, as described in [27, 29], involves using single-neuron approximation methods for initial verification, shifting to multi-neuron approximation methods upon failure. While pragmatically appealing, this strategy does not fully exploit the potential synergies between these two approximation methods yet, suggesting an opportunity for a more effective verification paradigm

This Work. This work bridges the gap. The idea is to *lift single-neuron approximations into single multi-neuron approximations with precision gain*. To this end, we formulate the approximation transition as a novel problem, called *Convex Approximation Lifting* (CAL), and develop a constructive approach, *Support Triangle Machine* (STM), to solving it. Beyond DNN verification, our approach has the potential to facilitate the approximation construction process in a wide range of applications and we expect it to catalyze further research in related fields. We brief more details below.

Lifting single-neuron approximations into multi-neuron ones with precision gain is very challenging. It essentially requires us to approximate a function in high-dimensional space using its approximations in lower-dimensional spaces. To better study the transition, we formulate a novel problem, CAL, to characterize the lifting process in a mathematical manner. Roughly speaking, a CAL problem considers a tuple $(X, F, \mathcal{S})$ and aims to derive a tight multi-dimensional approximation (corresponds to the multi-neuron approximation) $\mathcal{M}$ for function F on domain X, utilizing several single-dimensional approximations (corresponds to the single-neuron approximations) $\mathcal{S}$. The rigorous definition and more details can be found in Sect. 2.

Given a CAL problem, a spectrum of solutions that vary in precisions[1] and computational costs exist. For instance, both convex hull (*Conv*), which offers the highest precision but is computationally expensive, and Cartesian product (*Cart*), which is extremely efficient but provides no precision gain, are typical CAL solutions. Unfortunately, neither of them is of our interest, as they do not consider the trade-off between precision and cost. Instead, a good solution should strike a balance between the two aspects, and apparently, finding such a solution is non-trivial.

STM addresses this challenge by exploiting the compositionality of single-neuron approximations and leveraging a special liftable structure. Specifically, STM is founded on two core insights: *(i) there exists a simple geometric structure, named the support triangle, along with an efficient triangle lifting technique that bridges single-dimensional with multi-dimensional approximations,* and *(ii) practical single-dimensional approximations can be decomposed into multiple components, each of which is liftable via the triangle lifting technique.*

Given a CAL problem, STM incrementally lifts each output coordinate through approximation decomposition and triangle lifting. Specifically, STM extends one output coordinate at a time. For each coordinate, it decomposes the single-dimensional approximation into several linear components, within which support triangles can be identified and the triangle lifting technique can be applied. Once all components are lifted, their solutions are combined to form an intermediate multi-neuron approximation, which then serves as the input for the next coordinate. By repeating this process across all output coordinates, STM ultimately produces the final multi-neuron approximation.

[1]Here 'precision' refers to the degree to which the over-approximation closely matches the geometry and boundaries of the convex hull. It quantifies the overall closeness between the over-approximated shape and the target hull.

Lastly, we prove the correctness of STM theoretically and evaluate its performance empirically. On a variety of CAL problems and DNN verification tasks, our experimental results demonstrate that STM not only applies broadly but significantly enhances precision compared to the Cartesian product. When benchmarked against PRIMA, the leading multi-neuron method in DNN verification, STM matches its precision gains while offering broader applicability and better efficiency.

Main Contributions. We make the following key contributions in this work:

- We propose to bridge the gap between single-neuron and multi-neuron approximation methods through lifting. We formulate the approximation lifting process as a CAL problem.
- We introduce the support triangle structure with an efficient triangle lifting technique that connects single-dimensional and multi-dimensional approximations together.
- We develop STM, a constructive approach to solving CAL problems. STM balances the trade-off between the lifted precision and computational cost.
- We establish the correctness of STM theoretically and evaluate its performance empirically on a variety of CAL problems and DNN verification tasks.

Paper Outline. The remainder of this paper is organized as follows. Sect. 2 formalizes the CAL problem. Sect. 3 presents the support triangle and the triangle lifting technique. Sect. 4 introduces STM. Sect. 5 details the evaluation of STM. Sect. 6 surveys related research and Sect. 7 concludes.

Notations. The following notations are used throughout this paper:

- We reuse the set intersect operator '$\cap$' to denote conjoining operations for constraints. When the context allows, the alphabet for constraints may be omitted for simplicity. For example, if $X = \{x > 0\}$, then $X \cap \{x < 1\}$ results in $\{x > 0,\ x < 1\}$ where the alphabet is $\Sigma = \{x\}$. When conjoining $X = \{x > 0\}$ with $Y = \{y < 1\}$, the result is $\{x > 0,\ y < 1\}$ and the corresponding alphabet is $\Sigma = \{x, y\}$.
- We denote an m-piecewise function F as $\langle s_0, F_1, \cdots, s_{i-1}, F_i, s_i, \cdots, F_m, s_m\rangle$, where F_i is a function defined on the interval $[s_{i-1}, s_i]$ and $F = F_i$ holds on that domain. When $s_0 = -\infty$ or $s_m = \infty$, we may omit s_0 or s_m, respectively. We say the function F is continuous iff. $F_1(s_1) = F_2(s_1), \cdots, F_{m-1}(s_{m-1}) = F_m(s_{m-1})$ and all F_i ($i \in [1, m]$) are continuous.

2 Convex Approximation Lifting

This section first recalls function over-approximations and then formally defines the CAL problem.

2.1 Function Over-approximation

A function can be geometrically visualized as a graph in its input-output space, where the input space is referred to as its domain. An over-approximation of a function acts as as an 'envelope' that encloses this graph. This envelope encapsulates all possible outputs the function may produce for inputs within the domain. A convex over-approximation impose the additional requirement that the envelope must be convex. For a given function, the most precise over-approximation is the convex hull of all its input-output pairs. However, constructing the convex hull in high-dimensional spaces is often computationally prohibitive, making further approximations necessary.

Scalable DNN verification approaches typically rely on approximating non-linear activation functions in high-dimensional spaces. These approximations are generally performed by either completely or partially ignoring the relationships among neurons. The former, known as *single-neuron approximation methods*, are simpler and more efficient but often lack precision. In contrast, the latter, *multi-neuron approximation methods*, offer higher precision but are complex and challenging to design and implement. This work aims to bridge the gap by constructing more precise multi-neuron approximations based on less precise single-neuron approximations.

2.2 Problem Formulation

Let $X \subset \mathbb{R}^n$ be a closed convex set, and let X_i denotes its projection onto the *i-th* coordinate axis. Let $F : \mathbb{R} \to \mathbb{R}$ be a single-valued unary function that can be extended in two principal ways:

1) To vector variables: $F(\vec{x}) = \big(F(x_1), \ldots, F(x_n)\big)$ for $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$;
2) To set variables, i.e. $F(S) = \{F(x) \mid x \in S\}$, where $S \subset \mathbb{R}$.

A set $Q \subset \mathbb{R}^{2n}$ is said to be an over-approximation of F on X if $\big(X, F(X)\big) \subset Q$. To align with terminology in neural network verification, we refer to Q as a *single-dimensional approximation* when $n = 1$, and as a *multi-dimensional approximation* when $n > 1$. We use the terms *single-dimensional (or multi-dimensional) approximation* and *single-neuron (or multi-neuron) approximation* interchangeably throughout this paper, to emphasize their relevance in mathematical and neural network verification contexts, respectively.

Definition 2.1 (Convex Approximation Lifting). Given a closed convex set $X \subset \mathbb{R}^n$ and a dimension-wise function $F : \mathbb{R}^n \to \mathbb{R}^n$, let $\mathcal{S}_i$ be a convex over-approximation of F on X_i for each $i \in 1, \ldots, n$. The *Convex Approximation Lifting* (CAL) problem $(X, F, \mathcal{S})$ is to find a multi-dimensional convex over-approximation $\mathcal{M}$ for $\big(X, F(X)\big)$.

While convex sets and convex approximations are general concepts, we assume they are represented as convex polyhedra in practice. We refer to the problem of lifting the *d-th* coordinate as a 1-D CAL problem, denoted as $(X, F, \mathcal{S}_d, d)$. Since a typical single-dimensional approximation $\mathcal{S}_d$ can be specified via two bounding functions, $F_d^{\leq}$ (upper approximation or upper bound) and $F_d^{\geq}$ (lower approximation or lower bound), we name $(X, F, F_d^{\geq}, d)$ as a 1-D upper-CAL problem and $(X, F, F_d^{\leq}, d)$ as a 1-D lower-CAL problem.

To better understand CAL, it is helpful to draw an analogy to 3-D reconstruction, a classical problem in computer graphics that involves assembling 3-D representations from 2-D images. Both problems *construct high-dimensional objects, i.e. $\mathcal{M}$, based on their projections, i.e. $\mathcal{S}$, to lower-dimensional space*. However, they differ in both target spaces (3-D vs. 2*n*-D) and underlying goals. CAL focuses on approximations and operates in potentially much higher dimensional space, while 3-D reconstruction aims to create visually accurate 3-D models from 2-D images.

2.3 Discussion

A critical aspect of the CAL problem is the introduction of single-dimensional approximations $\mathcal{S}$. Although it is possible to derive $\mathcal{M}$ directly without using $\mathcal{S}$ – as seen in works like PRIMA [29] and WraLU [27] – such approaches are often complex and difficult to design. By incorporating $\mathcal{S}$, we split the construction of $\mathcal{M}$ into two steps: (1) constructing $\mathcal{S}$ for each coordinate; and (2) lifting them up to obtain $\mathcal{M}$. This design not only simplifies the construction of multi-dimensional approximations but also allows researchers to improve each step separately.

One might consider using the convex hull, i.e. $Conv\big(X_i, F(X_i)\big)$, as the single-dimensional approximation $\mathcal{S}_i$. This is, however, not always practical due to the restriction to polyhedral forms. There are cases where the convex hull contains too many hyperplanes or can not represented as a polyhedron at all – for example, the convex hull of e^x on any non-empty interval. By assuming $\mathcal{S}$ is given, we not only avoid such difficulties, but also generalize the problem, enabling the exploration of a broader range of convex approximations beyond convex hulls.

Example 2.2. Let F be the *ReLU* function. It we take the triangle approximation, i.e. the most precise single-neuron approximation, as S_i, then the CAL problem is equivalent to the *ReLU* hull approximation problem [27]. Specifying S_i using other abstractions, such as the box abstraction, symbolic interval [39] or DeepPoly [35], yields a variety of *ReLU* approximation problems. Solving these results diverse multi-neuron *ReLU* approximations.

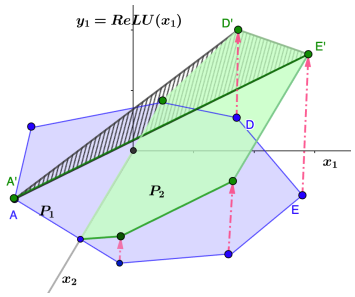


Fig. 1. A conceptual illustration of WraLU lifting. In the figure, points A and A' collapse together.

3 Support Triangle

This section recalls convex polytopes and the WraLU method, and introduces support triangles.

3.1 Convex Polytope

A convex polytope can be described in two primary ways: as the convex hull of its extremal points (the V-representation) or as the intersection of a set of linear constraints (the H-representation). Transforming the V-representation to the H-representation is known as the *convex hull problem*, while the reverse is referred to as the *vertex enumeration problem*. Both problems are NP-hard.

In high-dimensional geometry, a k-face L of an n-dimensional polytope P is defined as a k-dimensional subset $L \subset P$ that satisfies exactly $(n-k)$-linearly independent constraints with equality. A 0-face is called a *vertex* and a $(k-1)$-face is known as a *facet*[14]. When $n = 2$, a 1-face is commonly referred to as an *edge*. In the special case where $n = 1$, the notions of facet and vertex coincide, and the convex polytope reduces to a line segment. This segment has two endpoints, each serving as both a vertex and a facet. Moreover, in any n-dimensional polytope, each $(n-2)$-face is shared by exactly two facets, illustrating the interconnected structure of polyhedral geometry.

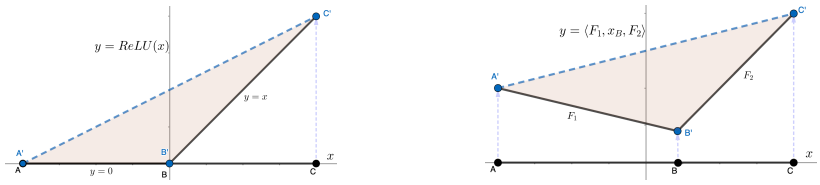
3.2 The WraLU Method

WraLU [27] is a convex hull approximation method for *ReLU* functions. It constructs a convex polytope that 'wraps' the convex hull by identifying the lower faces and assembling their adjacent faces[2]. The core idea of WraLU is to determine *an $(n+1)$-d hyperplane by an n-d hyperplane and a point out of it*, which we refer to as *Hyperplane-Point Pairing (H-P Pairing)* technique in this work.

Example 3.1. Let X be a 2-D polytope and $Y = ReLU(X)$. Fig. 1 visualizes how WraLU approximates the *ReLU* hull in the (x_1, x_2, y_1) space. Here, the purple area denotes X and the pink dashed arrows represent the lifting operations induced by *ReLU*. For instance, point E lifts to E'. Upon the *ReLU* transformation, X is split into two parts: P_1 (on the left, where $x_1 \leq 0$) which remains unchanged, and P_2 (on the right, where $x_1 \geq 0$) which is lifted up to form the green area. The *ReLU* hull is the smallest convex polytope containing both P_1 and P_2, which is $Conv(P_1, P_2)$.

Computing the convex hull in high dimensional space can be very costly, and thus WraLU approximates it instead. It constructs a polytope by identifying and locating the 'missing' faces (the faces of the convex hull, except P_1 and P_2). This is done by identifying open edges of P_1 and P_2 and pairing them with lifted vertices. For example, the open edge $\overline{D'E'}$ is paired with vertex A', forming the shadowed triangle $\Delta A'D'E'$. Together, all these faces complete the polytope and form an upper wrapper for the convex hull. With this, WraLU can continue to extend the coordinate y_2. □

[2]This construction approach was internally referred to as *Selective Faces Assembly* (SFA), reflecting the process of how the polytope is constructed.



(a) (b)

Fig. 2. Generalization of WraLU method.

3.3 On the H-P Pairing Technique

A deeper exploration of the H-P Pairing technique reveals its potential for generalization, inspiring the discovery of the support triangle structure and an efficient lifting technique. Below, we detail two key observations.

3.3.1 Beyond Multi-Dimensional Approximations. A fact that might be overlooked is that the H-P Pairing technique can be used to construct single-dimensional approximations for *ReLU*.

Example 3.2. In Fig. 2a, suppose the input polytope is the interval $\overline{AC}$ in 1-D space, then $\Delta A'B'C'$ is the *ReLU* hull, the most precise convex over-approximation of the *ReLU* function.

When applying the H-P Pairing technique, it first identifies two lower faces, i.e. line segments $\overline{A'B'}$ and $\overline{B'C'}$. Then, it enumerates open facets, which are 0-d hyperplanes, a.k.a. points, denoted as A' and C', along with two vertices A' and C'. Subsequently, it assembles new edges adjacent to $\overline{A'B'}$ and $\overline{B'C'}$, which are edges connecting the 0-*d* hyperplane A' with the vertex C' and connecting the 0-*d* hyperplane C' with the vertex A', both contributing to the line segment $\overline{A'C'}$. Together with two lower faces, we final obtain the triangle $\Delta A'B'C'$, which is exactly the *ReLU* hull. □

While single-dimensional approximations could be computed via other methods, our observation indicates *the H-P Pairing technique provides a unified way to approximate ReLU functions in both single-dimensional and multi-dimensional settings.* Such unification suggests us to investigate the nexus between these two approximations. Indeed, single-dimensional approximations serve as a guidepost for constructing multi-dimensional approximation of the *ReLU* function. Specifically, *the two sides, which encode the ReLU semantics, illustrate how the function transforms on the input polytope; the third side provides insights for pairing facets (or hyperplanes) and vertices.* It is this connection that makes the efficient and tight approximation lifting possible.

3.3.2 Beyond ReLU Hull Approximations. The H-P Pairing technique can be used to approximate functions beyond *ReLU*. While non-trivial for general functions, applying the H-P Pairing technique to continuous, 2-piecewise linear functions is straightforward. The only adjustment is to apply F instead of *ReLU* to capture the function's semantics. See the example below.

Example 3.3. Taking a 2-D polytope X as input, let's consider the function $F = \langle -\frac{1}{2}x + \frac{3}{2}, 1, x\rangle$, as shown in Fig. 2b, for lifting. To extend the y_1 coordinate, we first compute two lower faces: $P_1 : X \cap \{x_1 \leq 1, y_1 = -\frac{1}{2}x_1 + \frac{3}{2}\}, P_2 : X \cap \{x_1 \geq 1, y_1 = x_1\}$. Then, we identify the open facets Hs of P_1 and P_2, as well as their vertices Vs. Note that, when calculating Vs, we first compute the vertices of X and then applies F, instead of *ReLU*. Next, we construct the wrapping hyperplanes by pairing a facet $H \in Hs$ and a proper vertex $V \in Vs$. Again, these constructed hyperplanes, together with two lower faces, collectively form a polytope, which will be utilized to extend the y_2 coordinate. □

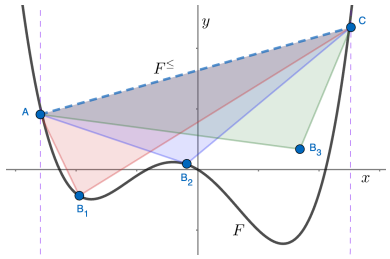


Fig. 3. Support triangles for a curved function.

3.4 Support Triangle

Now we are ready to introduce the *support triangle* structure and the triangle lifting technique.

Definition 3.4 (Support Triangle). Let F be a unary single-valued function on an interval I, and let $F^{\leq}$ be its upper approximation. A triangle $T = \Delta ABC$ is a *support triangle* for $F^{\leq}$ w.r.t. (F, I) iff.:

(i) $\forall (x, y) \in T,\ s.t.\ x \in I,\ F(x) \leq y \leq F^{\leq}(x)$;
(ii) $\forall x \in I,\ \exists y,\ s.t.\ (x, y) \in F^{\leq} \Leftrightarrow (x, y) \in \overline{AC}$.

In the above definition, condition (i) requires the triangle T to be positioned between F and $F^{\leq}$, and condition (ii) stipulates that one side of the triangle, i.e. $\overline{AC}$, coincides with $F^{\leq}$ over the interval I. We make the vertex order matter here in order to simplify representation; when denoting a support triangle as ΔABC, we assume without loss of generality $\overline{AC} = F^{\leq}$ over I and the x-value of A is smaller than that of C. Here, $\overline{AC}$ is termed the *supported facet*; $\overline{AB}$ and $\overline{BC}$ are *supporting facets*, with corresponding hyperplanes being termed *supported* and *supporting hyperplanes*, respectively.

Definition 3.5 (Stableness). Let ΔABC be a support triangle for $F^{\leq}$ w.r.t. (F, I). ΔABC is a *stable support triangle*, if there exists at least one point on $\overline{AB}$ other than A and at least one point on $\overline{BC}$ other than C that lie on function F; otherwise, it is an unstable support triangle. Moreover, if B lies on F, ΔABC is a *Type I stable support triangle*; otherwise, it is a *Type II stable support triangle*.

Example 3.6. In Fig. 2a, $\Delta A'B'C'$ is a support triangle for $\overline{A'C'}$ w.r.t. *ReLU* over $\overline{AC}$. Similarly, in Fig. 2b, $\Delta A'B'C'$ is a support triangle for $F^{\leq} = \overline{A'C'}$ w.r.t. $F = \langle -\frac{1}{2}x + \frac{3}{2}, 1, x \rangle$ over $\overline{AC}$. Fig. 3 further shows three example support triangles ΔAB_1C, ΔAB_2C, ΔAB_3C for the curved function. The first two are (Type I) stable support triangles and the last one is an unstable support triangle. □

Notably, when ΔABC is a support triangle for $F^{\leq}$ w.r.t. (F, I), any triangle $\Delta AB'C$ with $B' \in \Delta ABC$ is also a support triangle for $F^{\leq}$ w.r.t. (F, I). In general, we have the following theorem.

Theorem 3.7. *Given a unary single-valued function F and its upper approximation $F^{\leq}$ over the interval I, there exists either zero or an infinite number of support triangles for $F^{\leq}$ w.r.t. (F, I).*

Proof. (Sketch) If the approximation $F^{\leq}$ is non-linear, it is easy to see that no support triangle exists. Conversely, at least one support triangle ΔABC exists, in which case an infinite number of support triangles can be constructed, e.g., by replacing vertex B with any point B' within ΔABC. □

Triangle Lifting Technique. Given a support triangle for an arbitrary approximation $F^{\leq}$ w.r.t. (F, I), we can approximate the lifted convex hull for a convex polytope X using the triangle lifting technique. Algo. 1 shows the procedure, which takes X, ΔABC_d, and the dimension d as inputs. First, it uses the supported face $\overline{AC}$ to form an upper approximation U_0. Then it computes the supporting faces P_1 and P_2, followed by extracting the open facets and vertices of the supporting

Algorithm 1: Triangle Lifting Algorithm

Procedure `TriangleLift` *(X, ΔABC_d, d)*
- Extract function $F = (F_1, s_1, F_2)$ from ΔABC_d
- // $\overline{AC}$ is the supported face
- $U_0 \leftarrow y_d \leq \overline{AC_d}$
- // Compute the supporting faces
- $P_1 \leftarrow X \cap \{x_d \leq s_1\} \cap \{y_d = F_1(x_d)\}$
- $P_2 \leftarrow X \cap \{x_d \geq s_1\} \cap \{y_d = F_2(x_d)\}$
- // Extract open facets and vertices of supporting faces
- $(H_1s, H_2s) \leftarrow ListHyperplanes(P_1, P_2) \setminus \{x_d = s_1\}$
- $(V_1s, V_2s) \leftarrow EnumerateVertices(P_1, P_2)$
- // Construct supported hyperplanes, get upper constraints for y_d
- Assemble supported hyperplanes Ps via (H_1, V_2) or (H_2, V_1)
- $U_s \leftarrow \bigcap_{P \in P_s} (y_d \leq P)$;
- $U_d \leftarrow U_s \cap U_0$
- **return** U_d

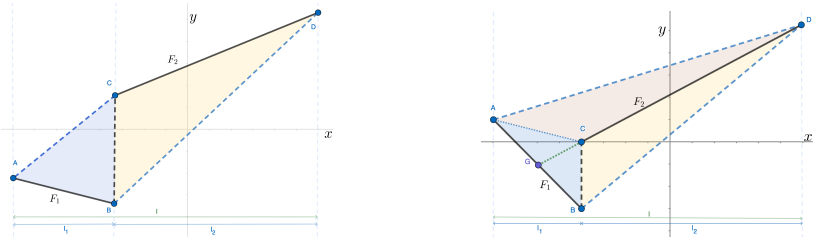
faces. Next, it constructs the supported hyperplanes and obtains the upper constraint set U_s for the d-th output coordinate. Finally, the upper constraints U_0 and U_s are combined and returned as the final upper approximation U_d.

Example 3.8. It is not hard to apply the triangle lifting technique to construct multi-dimensional approximations. For instance, to lift functions such as *ReLU* or any 2-piecewise linear, continuous function, we can iterative invoke function `TriangleLift` for each output coordinate. A sample procedure can be found in Algo. 2, where ΔABC^n encodes n single-dimensional approximations and $\mathcal{S}_i = \Delta ABC_i$. The function `FullDimsLift` iterates over all output coordinates, and for each output coordinate, it invokes function `TriangleLift` to compute the upper approximation U_d for the *d-th* output coordinate, and uses $y_d \geq \overline{AB_d}$ and $y_d \geq \overline{BC_d}$ as the lower approximation L_d, then conjoin them to compose $\mathcal{M}_d$ accordingly. Once all output coordinates are processed, the final multi-dimensional approximation $\mathcal{M}_n$ can be obtained. □

Algorithm 2: A sample CAL procedure via triangle lifting technique

Procedure `FullDimsLift` *(X, ΔABC^n)*
- $\mathcal{M}_0 \leftarrow X$
- **foreach** $d \in [1...n]$ **do**
  - $U_d \leftarrow$ `TriangleLift`$(\mathcal{M}_{d-1}, \Delta ABC_d, d)$
  - $L_d \leftarrow (y_d \geq \overline{AB_d}) \cap (y_d \geq \overline{BC_d})$
  - $\mathcal{M}_d \leftarrow U_d \cap L_d$
- **return** $\mathcal{M}_n$

Degenerate Case. When all vertices of a support triangle are co-linear, we encounter the degenerate case where the support triangle is a line segment, termed a *support line segment*. Most properties about support triangles discussed in this work apply to support line segments. In particular, the triangle lifting technique can be applied, but the lifted approximation is, unsurprisingly, of the same precision as the Cartesian product. Indeed, *the triangle lifting technique degenerates to the Cartesian product in the worst case* and *the triangle lifting technique is a natural generalization of the Cartesian product*, both conforming to our intuition.



(a) (b)

Fig. 4. Non-continuous 2 piecewise functions handling.

3.5 On Specific CAL Problems

The following shows how specific CAL problems can be solved through the triangle lifting technique. In particular, we consider the CAL problem $(X, F, \mathcal{S})$ where F is a piecewise function with two linear pieces and $\mathcal{S}_i$ is the convex hull of F on X_i. Based on F's continuity, we have two possibilities.

3.5.1 Continuous Functions. A continuous function with two linear pieces can be either concave or convex. As concave functions can be turned into convex functions by negation, thus only the convex case is discussed below. In fact, for convex continuous functions with two linear pieces, the upper approximation can be lifted by applying the triangle lifting technique as shown in Sec. 3.3.2. For lower approximation lifting, we need to split it into two sub-problems at the non-differential point of F, and then identify a support line segment for each sub-problem. The triangle lifting technique can be applied to each sub-problem, and the final approximation is the conjoining of the two lifted approximations. Conjoining the lifted upper and lower approximations leads to an intermediate multi-dimensional approximation $\mathcal{M}_d$.[3]

It's worth to note that, applying the Cartesian product to all single-dimensional lower approximations yields the same approximation. This is not a surprising result, as the *triangle lifting technique degrades to the Cartesian product when support triangles degrade to support line segments.*

3.5.2 Non-Continuous Functions. For a non-continuous function $F = (F_1, s_1, F_2)$, where $F_1(s_1) \neq F_2(s_1)$, there arise two cases. Specifically, suppose the domain is the interval I ($s_1 \in I$), we first extend F_1 and F_2 to the whole domain of I as F_1^e and F_2^e, and then based on whether they intersect within the interval I, two cases need to be considered:

(i) F_1^e and F_2^e do not intersect, then the convex hull would be a quadrangle and both functions contribute the edges of the hull, as Fig. 4a;
(ii) F_1^e and F_2^e intersect, the convex hull would be a triangle and either of F_1 or F_2 is completely enclosed, as Fig. 4b.

In case (i), the upper and lower approximations are symmetric and thus we only consider how the upper approximation is lifted here. In this case, we decompose the lifting problem into two sub-problems, based on the split point s_1. For the F_1 part over I_1, we identify a support triangle ΔABC, where $\overline{AC}$ is the upper approximation, and the triangle lifting technique applies. For the F_2 part over I_2, we identify a support line segment $\overline{CD}$, and then the Cartesian product applies. The final approximation is the conjoining of the two lifted approximations.

In case (ii), to lift the upper approximation $\overline{AD}$, we identify a support triangle ΔAGD, where G is the intersection point of F_1 and F_2^e, so the triangle lifting technique applies. It's worth to note

[3]The reason behind the conjoining operation will be explained in Sec. 4.4.

that, ΔACD is also a support triangle, but not as good as ΔAGD due to the line segment $\overline{AC}$ falling above $\overline{AG}$ and $\overline{GC}$. Differently, ΔABD is not a support triangle as $\overline{AD}$ falls below F_2. Furthermore, lifting the approximation via the triangle ΔABC and the line segment $\overline{CD}$ produces an invalid upper approximation, as $\overline{AC}$ falls within the convex hull of F. For lifting the lower approximation, it is similar with case (i) where two parts should be addressed separately and a support triangle and a support line segment can be identified.

By now, we have presented how to solve a restrictive set of CAL problems via support triangles. Along the text, we have also mentioned techniques such as decomposition and conjoining. Both provide a good foundation for understanding how general CAL problems can be solved via STM.

4 Support Triangle Machine

This section illustrates *Support Triangle Machine*, our approach to solving general CAL problems. After presenting an overview, we describe each component of STM in detail. We also discuss the correctness and complexity analysis of STM.

4.1 Overview of STM

The key idea of STM is to decompose a CAL problem into multiple simpler lifting problems where the triangle lifting technique can be applied and construct a solution to the original problem incrementally. Technically, STM addresses a CAL problem through:

- *incremental construction* — handling each output dimension sequentially to progressively build the multi-dimensional approximation; and
- *compositional lifting* — decomposing a 1-D CAL problem into multiple sub-problems, applying the triangle lifting technique, and synthesizing an intermediate approximation accordingly.

More precisely, given an arbitrary CAL problem $(X, F, \mathcal{S})$ where $X \subset \mathbb{R}^n$, STM incrementally constructs the multi-dimensional approximation $\mathcal{M}$, by extending one output dimension at a time. For each output dimension, d for example, the corresponding 1-D CAL problem is separated into two independent sub-problems, i.e. a 1-D upper CAL problem and a 1-D lower CAL problem. STM addresses each of them in a compositional manner. In specific, it breaks the sub-problem into several atomic components, based on the linearity of $\mathcal{S}_d$. Next, for each atomic problem, a support triangle can be identified and the triangle lifting technique can be applied. Once all the atomic problems are solved, the constraints of all the lifted approximations can be combined together to produce an intermediate multi-dimensional approximation up to dimension d, i.e. $\mathcal{M}_d$. This process iterates until $d = n$, where the final approximation $\mathcal{M}_n$ is constructed. A formalized algorithm is shown in Algo. 3. Below we illustrate how a 1-D approximation is lifted using a conceptual example.

Example 4.1. Fig. 5 shows a function F and its over-approximation $ACEFGIJ$ in form of a polygon. The polygon can be divided into an upper part ACE and a lower part $FGIJ$. Below, we consider how to lift an polytope X based on these approximations.

By invoking procedure `OneDimApproxLift`, we first split the lifting problem into two sub-problems, one focusing on the upper approximation $F^{\leq}$ and the other on the lower approximation $F^{\geq}$. Taking the upper approximation as example, $F^{\leq}$ consists of two linear parts, $F_1^{\leq}$ and $F_2^{\leq}$ (line segments $\overline{AC}$ and $\overline{CE}$). The procedure `OneDimUpperApproxLift` decomposes the upper lifting problem into two atomic lifting problems, where one support triangle (illustrated by the red and yellow triangles, i.e. ΔABC and ΔCDE) can be identified for each of them. The triangle lifting technique is then applied, and the results U_1 and U_2 are combined to form the lifted upper approximation U. The process for lifting lower approximation lifting problem follows a similar pattern but involves decomposing into three atomic lifting problems, due to $F^{\geq}$ having three linear parts, with only one support line segment existing for $\overline{FG}$. □

Algorithm 3: Support Triangle Machine

Procedure `OneDimUpperApproxLift` $(X, F, F^{\leq}, d)$
  Rewrite $F^{\leq}$ as $\langle s_0, F_1^{\leq}, s_1, F_2^{\leq}, s_2, \cdots, s_{m-1}, F_m^{\leq}, s_m \rangle$
  foreach $i \in [1...m]$ **do**
    Identify a support triangle T for $F_i^{\leq}$ w.r.t. $(F, [s_{i-1}, s_i])$
    $U_i \leftarrow$ `TriangleLift`$(X \cap \{x_d \geq s_{i-1}\} \cap \{x_d \leq s_i\},\ T,\ d)$
  $U \leftarrow \bigcap_{i=1}^{m} U_i$
  return U

Procedure `OneDimLowerApproxLift` $(X, F, F^{\geq}, d)$
  // Similar to procedure `OneDimUpperApproxLift`, omitted.

Procedure `OneDimApproxLift` $(X, F, \mathcal{S}_d, d)$
  Extract upper approximation $F^{\leq}$ and lower approximation $F^{\geq}$ from $\mathcal{S}_d$
  $U \leftarrow$ `OneDimUpperApproxLift`$(X,\ F,\ F^{\leq},\ d)$
  $L \leftarrow$ `OneDimLowerApproxLift`$(X,\ F,\ F^{\geq},\ d)$
  return U, L

Procedure `FullDimsApproxLift` $(X, F, \mathcal{S})$
  $\mathcal{M}_0 \leftarrow X$
  foreach $d \in [1...n]$ **do**
    $U_d,\ L_d \leftarrow$ `OneDimApproxLift`$(X, F, \mathcal{S}_d, d)$
    $\mathcal{M}_d \leftarrow U_d \cap L_d$
  return X

4.2 Problem Decomposition

STM considers decomposing a CAL problem into atomic lifting problems, which are defined below.

Definition 4.2 (Atomic Lifting). A 1-D upper CAL problem $(X, F, F^{\leq}, d)$ is atomic iff. $F^{\leq}$ is linear over X_d. Similarly, a 1-D lower CAL problem $(X, F, F^{\geq}, d)$ is atomic iff. $F^{\geq}$ is linear over X_d.

Given the single-dimensional approximation $\mathcal{S}$ is typically in form of convex polyhedra, its upper (or lower) part $\mathcal{S}^{\mathrm{u}}$ at any dimension d can be decomposed into multiple linear parts. In fact, $\mathcal{S}_d^{\mathrm{u}}$ is a continuous, concave, piecewise linear function, and can be rewritten as $\langle s_0, F_1^{\leq}, s_1, \cdots, s_{m-1}, F_m^{\leq}, s_m \rangle$, with $F_i^{\leq}(s_i) = F_{i+1}^{\leq}(s_i)$ ensuring continuity and $F_i^{\leq}$ being linear. This simple structure enables us to split the upper approximation lifting problem into m atomic sub-problems, each denoted as $({}^{j}M_{d-1}, F, F_j^{\leq}, d)$ where ${}^{j}M_{d-1}$ represents $\mathcal{M}_{d-1} \bigcap \{x_d \geq s_{j-1}\} \bigcap \{x_d \leq s_j\}$.

4.3 Atomic Lifting

In STM, each atomic lifting problem is addressed by identifying the support triangle and applying the triangle lifting technique. The first step is to determine whether support triangles exist:

Theorem 4.3. *Support triangles exist for an atomic lifting problem $(X, F, F^{\leq}, d)$ if and only if $F^{\leq}$ strictly bounds F on the open interval (X_d^l, X_d^u); that is $F(x) < F^{\leq}(x)$ holds for $\forall x \in (X_d^l, X_d^u)$.*

Proof. (Sketch.) If $F = F^{\leq}$ holds for some $x^* \in (X_d^l, X_d^u)$, then apparently no support triangles exist, as the point $(x^*, F(x^*))$ can not simultaneously lie above and below a triangle. □

In practice, when the approximation strictly bounds F, we can identify a support triangle ΔABC by locating vertex B, since A and C are the two endpoints of $F^{\leq}$. In fact, B can be any point that lie

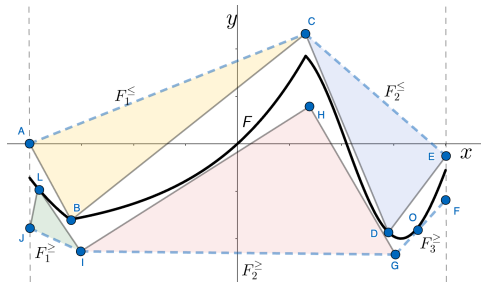


Fig. 5. Conceptual illustration of STM.

between F and $F^{\leq}$, as long as $\overline{AB}$ and $\overline{BC}$ do not intersect F. There are typically infinitely many such positions for B, leading to infinitely many possible support triangles, as formalized in Theorem 3.7.

However, different support triangles can lead to different levels of lifted precision. The challenge, therefore, is to identify the optimal support triangle. Here, 'optimal' refers to the triangle that yields the most precise lifted approximation. Unfortunately, finding such a triangle requires considering the three elements $X, F, F^{\leq}$ collectively, which can be highly complicated and potentially intractable. To address this, we choose to narrow the search space first.

Theorem 4.4. *Given two support triangles T_1 and T_2, if $T_1 \supseteq T_2$, then the approximation lifted by T_1 through triangle lifting technique is at least as precise as that obtained from T_2.*

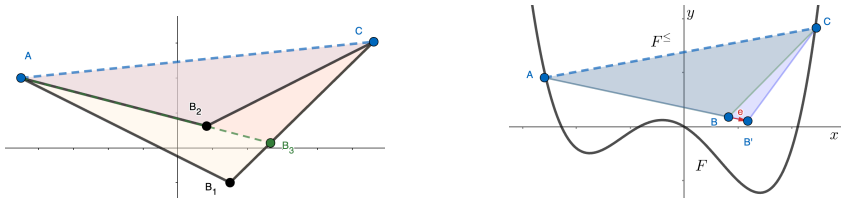
Proof. Consider an upper approximation lifting problem where $T_1 = \Delta AB_1C$ and $T_2 = \Delta AB_2C$, as shown in Fig. 6a. Assuming $T_1 \neq T_2$, we know $\overline{AB_1}$ and $\overline{B_1C}$ locate below $\overline{AB_2}$ and $\overline{B_2C}$ respectively. We extend $\overline{AB_2}$ to intersect with $\overline{B_1C}$ at point B_3 and it is easy to see $T_3 = \Delta AB_3C$ is also a support triangle. To prove the theorem, we only need to prove (i) the approximation lifted by T_3 is at least as precise as that by T_2 and (ii) the approximation lifted by T_1 is at least as precise as that by T_3.

Regarding (i), we divide the input X into two regions, i.e. $X_i = X \cap \{x_d \geq x_{B_2}\} \cap \{x_d \leq x_{B_3}\}$ and $X_{-i} = X \backslash X_i$ and we have three cases. (1) For points and hyperplanes falling in X_{-i}, the pairing relationships are the same when lifting using T_2 and T_3. Since $\overline{CB_3}$ locates below $\overline{CB_2}$, each lifted H or lifted V in a pairing relation by T_3 is lower than that by T_2, and thus the constructed faces by T_3 is lower than that by T_2. (2) For points and hyperplanes falling in X_i, the constructed faces via T_3 always fall below the hyperplane $\overline{B_2C}$ while those by T_2 may fall above the hyperplane $\overline{B_2C}$, and thus we have the similar property. (3) For hyperplanes crossing the boundary of X_i and X_{-i}, we can split them into two parts and apply the above analysis to each part accordingly. Therefore, we can safely conclude the approximation lifted by T_3 is more precise than or as precise as that by T_2. Till now, we have proved (i). Similarly, we can prove (ii). Combining (i) and (ii), the theorem holds. □

The theorem suggests that the lifted approximation by T_2 encloses the one by T_1, ensuring that T_1 offers no less precision than T_2. We further have the following:

Theorem 4.5. *Given an atomic lifting problem $(X, F, F^{\leq}, d)$ $\big($or $(X, F, F^{\geq}, d)\big)$, if support triangles exist, then there exists at least one stable support triangle that is optimal.*

Proof. (Sketch.) Assume, for contradiction, that all optimal support triangles are unstable, and let ΔABC be one such triangle. Since ΔABC is unstable, vertex B does not lie on the function F. As illustrated in Fig. 6b, extend the edge $\overline{AB}$ to a new point B' such that $|BB'| = \epsilon$, where $\epsilon > 0$ is arbitrarily small, and $\overline{B'C}$ still does not intersect F. Then $\triangle AB'C$ is also a support triangle. Moreover, since $\triangle ABC \subset \triangle AB'C$, the lifted approximation by $\triangle AB'C$ is at least as precise as that by $\triangle ABC$. Hence, $\triangle AB'C$ is also optimal.



(a) (b)

Fig. 6. Conceptual demonstration for proofs.

By renaming the point B' as B, we can repeat this process until either B' lies on F, or $\overline{B'C}$ starts to intersect F. In either case, we obtain a stable support triangle $\Delta AB'C$ that is optimal, which contradicts the assumption that no optimal support triangle is stable.

Therefore, we can conclude at least one optimal support triangle must be stable. □

Considering the above theorems, we introduce an effective triangle selection heuristic which *chooses the support triangle with the largest area*. The intuition is that a larger triangle is likely to cut off more volume during lifting, indicating a potential for higher precision. While no optimal guarantee can be made, our evaluation in Sec. 5.2.1 shows that this heuristic is effective in practice.

4.4 Approximation Composition

Once all atomic lifting problems are resolved, an intermediate multi-dimensional upper approximation can be composed, by aggregating all lifted approximations, i.e. $\bigcap_{j=1}^{m} \texttt{AtomicLift}({}^{j}M_{d-1}, F, F_j^{\leq})$. The theorem below consolidates the correctness of this construction.

Lemma 4.6 (Compositionality). *For a* 1*-D approximation lifting problem* $(X, F, \mathcal{S}_d^u)$*, the upper approximation* U *yielded by the* d*-th iteration of STM upper-approximate* $Conv\big(X, f(X_d)\big)$.

Proof. Suppose STM splits $\mathcal{S}_d^{\mathrm{u}}$ into m_d linear parts, resulting in m_d atomic lifting problems. We apply induction to show that the union of all lifted upper approximations constitutes an upper approximation of the original problem.

For $m_d = 1$, the lemma's correctness is ensured by the construction principles of WraLU [27].

For $m_d = 2$, the problem is decomposed into two atomic problems, each with one support triangle as Fig. 5. Denoting supporting hyperplanes from left to right as ①, ②, ③, ④ (as $\overline{AB}$, $\overline{BC}$, $\overline{CD}$, $\overline{DE}$) and supported hyperplanes as $\mathcal{L}$ and $\mathcal{R}$ (as $\overline{AC}$, $\overline{CE}$), we need to prove $\mathcal{L}$ is an upper approximation of all four pieces, i.e. ①, ②, ③, ④, then $\mathcal{R}$ can be proved similarly. Since $\mathcal{L}$ is an upper approximation of ① and ②, we only need to show it also upper-approximate ③ and ④. The following proof considers how convex hulls are formed and leverages the convexity of X and $\mathcal{S}_d$.

Before proving, we introduce some concepts to describe the observability between points on supporting hyperplanes. Assuming we only consider upper approximations, two points on supporting hyperplanes are *observable* if they can be connected by a line segment without intersecting any other supporting hyperplanes. These points are deemed *upper-observable* if the connecting line segment lies above (or on) the supporting hyperplanes, and *lower-observable* if it lies below (or on) these planes. For instance, in Fig. 5, points A, B, C are mutually upper observable; A is not observable by D or E; B and C are both upper- and lower-observable, while B and D are lower-observable.

Now we prove by contradiction. Suppose $\mathcal{L}$ divides vertices of $Conv(③, ④)$ into halves and K is the highest vertex above $\mathcal{L}$. Let us consider constructing an upper approximation of $\mathcal{L}$ and K. We know all the vertices of $\mathcal{L}$ are lifted by ① or ②, and K is lifted by ③ or ④. The tightest upper

approximation must have at least one additional edge that connects K with one vertex, say J, (other than the vertex with the same x value as the split point s) of $\mathcal{L}$. A necessary condition of such a connection is the two vertices are upper-observable. Below, we show this is not possible.

(1) If J and K are lifted by non-adjacent hyperplanes (i.e., ① and ③), they are not upper-observable. Assuming a vertex J lifted from J_0 by ① can observe a vertex K lifted from K_0 by ③, then we can draw the line segment $\overline{J_0K_0}$, and we know $\overline{J_0K_0} \subset X$ due to the convexity of X and it must intersect with zone ②. After lifting, $\overline{JK}$ still intersects hyperplane (not the facet) ② and, more importantly, the intersection point must be lower than $\cap$ (point E in the figure), otherwise, it contradicts the convexity of $\mathcal{S}_d$. For the the intersection point, there are two cases here. First, if it is within the facet ②, then J and K are not observable. Second, if it is below facet ②, then they are only lower-observable. Thus, points on non-adjacent supporting hyperplanes are not upper-observable.

(2) If J and K are lifted by adjacent hyperplanes (i.e., ② and ③), it is not hard to see that they are not upper-observable.

Therefore, such K does not exist and thus $\mathcal{L}$ is an upper approximation of all the four pieces. So is $\mathcal{R}$. Therefore, $\mathcal{L} \cap \mathcal{R}$ is a valid upper approximation of all four pieces.

For $m_d > 2$, we first prove $m_d = 3$ case, assuming the regions are A, B, C. Based on $m = 2$ case, we know the upper approximation of A upper-approximates B. Similarly, it upper-approximates C. So, it upper-approximates all three regions. Similarly, we can prove for B and C, and thus conjoining them establishes a valid upper approximation of all the regions. For $m_d > 3$, we can prove that upper approximation of any region also upper-approximate other $m - 1$ regions. Hence, conjoining all together establishes a valid upper approximation of all the regions. □

Intuitively, the lemma states that the constraints of all the upper approximations for decomposed problems can be conjoined together, without concerning whether the convexity breaks. The same logic applies to lower approximations. Hence, we assert the following theorem:

Theorem 4.7. *The approximation produced by STM is an over-approximation of* $\big(X, f(X)\big)$.

Proof. We first prove that the upper approximation yielded by STM is an upper approximation of $\big(X, f(X)\big)$. When $d = 0$, $\mathcal{M}_0 = X$ is a trivial upper approximation of X. Assume that the property holds for d dimensions, then we know it holds for $d + 1$ dimension based on Lemma 4.6. Hence, through induction, we know $\mathcal{M}_n^u$ (when $d = n$) upper-approximates $(X, f(X))$.

The same logic applies to the lower approximation. Combining both validates the theorem. □

4.5 Complexity Analysis

We now analyze the asymptotic complexity of STM. The building block of STM is the H-P pairing technique, which is used to solve an atomic lifting problem. Lifting an atomic problem $(X, F, F_d^{\leq}, d)$ (or $(X, F, F_d^{\geq}, d)$) has complexity $O(H_d \cdot V_d)$, where H_d and V_d denote the number of constraints and vertices, respectively. To solve a 1-D approximation lifting problem $(X, F, \mathcal{S}_d, d)$, STM applies the H-P pairing technique m_d times, where m_d is the number of linear segments in $\mathcal{S}_d$. Thus, the complexity becomes $O(m_d \cdot H_d \cdot V_d)$. For a CAL problem $(X, F, \mathcal{S})$, STM iteratively considers each coordinate, each of which constitute a 1-D approximation lifting problem. Therefore, the overall complexity is $\sum_{d=1}^{n} O(m_d \cdot H_d \cdot V_d)$, where n is the number of dimensions. Deriving a more compact or closed-form expression is hard in general, as no universal relationship exists between H_d and V_d.

In practice, two additional considerations may affect the overall cost. First, the input polytope X is typically given in H-representation only. Computing its V-representation requires a vertex enumeration procedure, which can be expensive in high dimensions. Second, the triangle lifting technique requires identifying a good support triangle. However, analytically determining the triangle may itself be computationally costly for certain functions.

5 Evaluations

This part presents our empirical evaluation of STM in solving CAL problems and verifying robustness property of DNNs. All experiments were conducted on a Ubuntu 22.04 desktop with an Intel Xeon 8 core 3.60GHz CPU and 64GB memory.

5.1 Implementations

We implemented STM as a prototype tool in Python. The implementation utilizes the `numpy` library for matrix operations and the `pycddlib` library for vertex enumeration.

Soundness. Numerical issues are a common challenge in floating-point computations, and improper handling can compromise the soundness of a technique. To consolidate our implementation, we performed a series of soundness tests to detect potential problems. Specifically, we randomly generated points within the input polytope X, and computed their exact images under the function F. We then checked whether these transformed points are enclosed within lifted approximation $\mathcal{M}$. Any point falling outside $\mathcal{M}$ indicates a potential bug or numerical error.

These tests were instrumental in identifying and addressing several issues, thereby enhancing the reliability of our implementation. Two particularly interesting numerical issues were discovered. (1) When a support triangle is excessively thin, lifting via the triangle can lead to unsound results. In such cases, we fallback to using the Cartesian product to ensure soundness. (2) When any supporting line, e.g. $\overline{AB}$ ($\overline{BC}$), is extremely steep, lifting vertices may introduce significant imprecision. To mitigate this, we slightly adjust the position of vertex B in ΔABC along $\overline{BC}$ (or $\overline{BA}$), thereby reducing the steepness of the supporting lines.

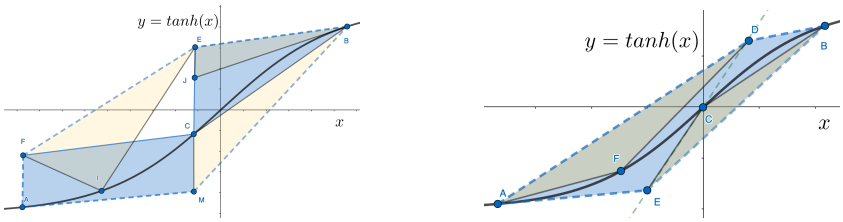
5.2 Convex Approximation Lifting

To evaluate the performance of STM on CAL problems, we selected a representative set of instances $(X, F, \mathcal{S})$, defined as follows:

- **Polytope** X: We generated 30 random polytopes with 8 constraints in 2-D space and another 30 polytopes with 27 constraints in 3-D space. Following [27], each variable is bounded within the interval $[-5, 5]$ to ensure the boundedness of X. For comparison with PRIMA, octahedral constraints were derived via preprocessing.
- **Function** F: We evaluated five representative functions, including LeakyReLU with varying α values (i.e. 0.01, 0.03, 0.05, 0.08, 0.1, 0.3, 0.5, 0.8), the square function x^2, the exponential function e^x, and two S-shaped functions: *Sigmoid* and *Tanh*.
- **Approximation** $\mathcal{S}$: We used both non-relational (e.g. boxes) and relational (e.g. polyhedra) abstractions as single-dimensional approximations. Further details are provided below.

Boxes. Boxes are simple yet widely used non-relational abstractions. Given the domain of F as $[l, u]$, the tightest box approximation of F is defined as $[l, u] \times [\min_{x \in [l,u]} F(x), \max_{x \in [l,u]} F(x)]$. While the box single-neuron approximation $\mathcal{S}$ is non-relational, our input polytope X and multi-dimensional approximation $\mathcal{M}$ remain relational.

Polyhedra. Polyhedra are a typical kind of relational abstractions using in program verification. For simplicity, we constrain each polyhedral approximation to use at most two linear constraints on both the upper and lower parts. For non-S-shaped functions, the upper approximation is the line segment connecting $(l, F(l))$ and $(u, F(u))$, while the lower approximation is determined by tangent lines at these endpoints. For S-shaped functions, we use two distinct polyhedral approximations: S_{PRIMA} (from PRIMA) and $S_{\varkappa}$ (developed in this work). These serve complementary purposes: S_{PRIMA} enables direct comparison with PRIMA, whereas $S_{\varkappa}$ showcases the capabilities of our method.



(a) S_{PRIMA} *approximation* (b) $S_{\mathcal{S}}$ *approximation*

Fig. 7. Single-dimensional approximations and example support triangles for *Tanh*.

- S_{PRIMA} splits the interval $[l, u]$ at a point c, chosen to minimize the area between the upper and lower bounds in the input-output plane. As shown in Fig. 7a, the two blue regions correspond to the original approximation in PRIMA. Since STM requires $\mathcal{S}$ to be convex, we use the convex hull of these regions – polygon $AFEBM$ – as the final approximation S_{PRIMA}.
- $S_{\mathcal{S}}$ forms a quadrilateral single-neuron approximation $ADBE$ as shown in Fig. 7b. The construction proceeds as follows: First, $\overline{DE}$ is set as the tangent line of F at $\big(0, F(0)\big)$[4]. Next, at point B, we draw a line with slope $k_b = \min\big(F'(x_B), k_{BC}\big)$, where $F'(x_B)$ is the slope of the tangent line of F at B, k_{BC} is the slope of segment $\overline{BC}$. This line intersects $\overline{DE}$ at D. A symmetric construction is applied to form $\overline{AE}$, resulting in a tight and sound polyhedral approximation.

Evaluation Metrics. We evaluated STM in terms of precision, efficiency and output complexity. Precision was measured by the volume of the lifted polytope (lifted volume). We uniformly sampled at least $1e5$ points in a bounding hypercube and estimated the volume by counting the proportion of points that fall within the polytope. Efficiency was measured by the runtime of each experiment. Output complexity was quantified by the number of constraints in the lifted polytope.

5.2.1 Triangle Selection Heuristic. The heuristic in Sec. 4.3 aims to identify a support triangle that significantly improves the lifted precision. To assess its effectiveness, we conducted experiments on atomic lifting problems – particularly 1D upper CAL problems where the approximation $F^{\leq}$ is linear. As a result, S-shaped functions like *Sigmoid* and *Tanh* were excluded, as their upper or lower approximation typically involve multiple linear segments.

Support Triangles. Fig. 8 demonstrates typical support triangles used in our evaluation. For LeakyReLU, ΔCAB in Fig. 8a represents the support triangle chosen by our heuristics for box upper approximation $\overline{CB}$, and ΔAOB for polyhedral upper approximation $\overline{AB}$. R is a random point on F, used to search for the optimal support triangle. Similarly, ΔCRB (or ΔARB) corresponds to an arbitrary support triangle for the box (or polyhedral) approximation. For the square function (in Fig. 8b), point D marks where the tangent line of F is parallel to $\overline{AB}$ and ΔADB is the support triangle selected by our heuristic for approximation $\overline{AB}$. Support triangles for the exponential function were determined similarly.

Quantitative Evaluation. To assess the triangle selection heuristic, we compute the ratio of the volume of the lifted approximation (lifted volume) via the triangle selected by our heuristic to the one via the optimal triangle, which is estimated through sampling. Specifically, for a fixed $F^{\leq}$, a Type I stable support triangle can be uniquely determined by its vertex on F. Thus, we collected stable support triangles by sampling 100 points evenly on F, and used the one with the minimal

[4]The idea of leveraging this tangent line in $\mathcal{S}$ was contributed by a student from Bai's group in the University of Queensland.

(a) (b)

Fig. 8. Single-dimensional approximations and example support triangles for LeakyReLU and square function.

lifted volume as the optimal support triangle. For comparison, we also report the average lifted volume, i.e. the mean volume of polytopes lifted by these 100 support triangles.

Table 1. Effectiveness of triangle-selection heuristics on atomic approximation lifting problems

	$X \subset \mathbb{R}^2$				$X \subset \mathbb{R}^3$			
$\mathcal{S}$	Boxes		Polyhedra		Boxes		Polyhedra	
$\mathbf{F}/\alpha$	Heuristic	Average	Heuristic	Average	Heuristic	Average	Heuristic	Average
0.01	1.566×	1.360×	1.399×	1.576×	3.214×	1.888×	1.261×	2.483×
0.03	1.529×	1.351×	1.403×	1.586×	2.987×	1.824×	1.265×	2.469×
0.05	1.473×	1.334×	1.389×	1.560×	2.812×	1.784×	1.264×	2.503×
0.08	1.460×	1.339×	1.399×	1.594×	2.661×	1.775×	1.265×	2.485×
0.10	1.415×	1.311×	1.416×	1.642×	2.551×	1.746×	1.271×	2.464×
0.30	1.271×	1.299×	1.420×	1.584×	1.588×	1.490×	1.268×	2.484×
0.50	1.135×	1.301×	1.426×	1.633×	1.165×	1.440×	1.282×	2.365×
0.80	1.023×	1.377×	1.385×	1.574×	1.022×	1.831×	1.284×	2.081×
Exp.	1.069×	1.330×	1.038×	1.215×	1.030×	2.037×	1.021×	1.349×
Sqr.	1.100×	1.301×	1.059×	1.241×	1.040×	1.571×	1.045×	1.472×

Results. Table 1 shows the statistical result, where '$k\times$' denotes the lifted volume is k times the optimal volume. From the table, we have the following observations. For LeakyReLU functions, when polyhedral approximations are employed, the triangles selected by our heuristic result in lifted volumes ranging from 1.26× to 1.43× the optimal volume. This means the heuristic's chosen triangle is at most 43% worse than the optimal one. In contrast, the average lifted volume ranges from 1.56× to 2.50× the optimal volume. For box approximations, our heuristic performs worse than the average case for LeakyReLU with small α values. For the exponential and square functions, our heuristic consistently outperforms the average case. These results suggest that the area is a crucial factor in optimal triangle selection. While not always being optimal, it performs effectively in most cases and generally surpasses average performance. Given its simplicity and efficiency, the heuristic is well-suited for practical use. Developing better methods to identify the optimal triangle remains an interesting and valuable direction for future work.

5.2.2 CAL Problem Solving. We now evaluate the performance of STM in solving CAL problems. Specifically, we compare it with the Cartesian product (*Cart*) and, where applicable, the approximation method in PRIMA and the convex hull (*Conv*) method.

For Non-S-Shaped Functions. We first assess STM's performance in solving CAL problems on non-S-shaped functions. All the configurations are the same as those in the previous part.

Table 2 presents the results. In the column headers, the notation $\mathcal{S}$-$\mathcal{L}$ (e.g., 'Box-Cart') denotes an approach that uses $\mathcal{S}$ for single-neuron approximation and $\mathcal{L}$ for the lifting strategy. The first

Table 2. Performance results on CAL problems with non-S-shaped functions (Time in ms.)

$X \subset \mathbb{R}^2$														
	Boxes-Cart		*Boxes-STM*			*Polyhedra-Cart*		*Polyhedra-STM*			*Conv*		
F/α	*Volume*	*Time*	*Volume*	*Time*	*#Cst.*	*Volume*	*Time*	*Volume*	*Time*	*#Cst.*	*Volume*	*Time*	*#Cst.*
0.01	979.428×	0.205	853.682×	0.573	26	2.538×	0.172	1.502×	0.324	18	1	3.480	15
0.03	1255.761×	0.211	1085.103×	0.602	26	2.744×	0.180	1.578×	0.342	18	1	3.585	15
0.05	2167.628×	0.209	1900.180×	0.607	26	2.738×	0.183	1.633×	0.341	18	1	3.611	15
0.08	1549.323×	0.211	1335.174×	0.600	26	2.656×	0.185	1.585×	0.336	18	1	3.623	15
0.10	2186.095×	0.212	1887.310×	0.602	26	2.658×	0.187	1.605×	0.338	18	1	3.614	15
0.30	2148.493×	0.208	1728.111×	0.588	26	2.597×	0.177	1.571×	0.330	18	1	3.488	15
0.50	5252.260×	0.198	3998.243×	0.566	26	2.717×	0.170	1.623×	0.310	18	1	3.062	15
0.80	36519.711×	0.192	25642.152×	0.550	26	2.617×	0.163	1.515×	0.306	18	1	3.199	15
Exp.	41.706×	0.268	12.833×	0.532	26	1.238×	0.148	1	0.303	20	-	-	-
Sqr.	11.410×	0.256	3.670×	0.501	24	1.369×	0.171	1	0.347	20	-	-	-
$X \subset \mathbb{R}^3$													
	Boxes-Cart		*Boxes-STM*			*Polyhedra-Cart*		*Polyhedra-STM*			*Conv*		
F/α	*Volume*	*Time*	*Volume*	*Time*	*#Cst.*	*Volume*	*Time*	*Volume*	*Time*	*#Cst.*	*Volume*	*Time*	*#Cst.*
0.01	944.621×	0.290	504.326×	0.520	42	7.410×	0.168	2.243×	0.289	29	1	364.753	148
0.03	955.104×	0.295	507.709×	0.529	42	7.334×	0.168	2.265×	0.300	29	1	361.956	148
0.05	967.748×	0.306	510.902×	0.537	42	7.171×	0.178	2.247×	0.294	29	1	362.071	148
0.08	1391.883×	0.311	717.284×	0.529	42	7.253×	0.170	2.243×	0.295	29	1	361.494	148
0.10	1299.271×	0.295	667.507×	0.524	42	7.315×	0.169	2.269×	0.292	29	1	361.177	148
0.30	5824.275×	0.290	2565.123×	0.531	42	7.381×	0.170	2.339×	0.295	29	1	356.367	148
0.50	26683.323×	0.290	9880.823×	0.526	42	7.689×	0.171	2.388×	0.294	29	1	317.431	148
0.80	394806.942×	0.291	111063.433×	0.518	42	6.040×	0.159	2.157×	0.284	29	1	348.489	148
Exp.	4173.622×	0.405	295.381×	0.811	42	1.469×	0.240	1	0.485	32	-	-	-
Sqr.	48.117×	0.334	5.375×	0.561	38	1.716×	0.207	1	0.399	32	-	-	-

Table 3. Performance results on CAL problems with S-shaped functions (Time in ms.)

$X \subset \mathbb{R}^2$																
	Boxes-Cart		*Boxes-STM*			S_{PRIMA}*-Cart*		S_{PRIMA}*-STM*			*PRIMA*			S_{∞}*-Cart*		S_{∞}*-STM*		
F	*Volume*	*Time*	*Volume*	*Time*	*#Cst.*	*Volume*	*Time*	*Volume*	*Time*	*#Cst.*	*Volume*	*Time*	*#Cst.*	*Volume*	*Time*	*Volume*	*Time*	*#Cst.*
Sigmoid	76.639×	0.308	50.768×	0.574	22	2.657×	0.301	2.344×	0.624	99.17	2.608×	1.072	38.73	1.029×	0.282	1	0.482	64.60
Tanh	36.300×	0.377	17.380×	0.654	22	3.051×	0.322	2.603×	0.700	97.60	2.972×	1.337	38.93	1.039×	0.316	1	0.823	61.00

column under each method reports the lifted volume, where '1' serves as the baseline, and entries marked as '$k\times$' indicate the relative volume compared to this baseline. We observe that STM consistently outperforms *Cart* in terms of precision. For LeakyReLU, STM achieves approximately a 20% precision gain on 2-D inputs and over 40% on 3-D inputs using box approximations. For polyhedral approximations, the improvements are even more substantial—exceeding 50% on 2-D inputs and 60% on 3-D inputs. Similar trends hold for the exponential and square functions, highlighting STM 's strength in lifting non-S-shaped functions with high precision. In terms of runtime, STM is slower than *Cart*, which is not surprising, yet remains highly efficient (under 1 ms) and offers a favorable precision-to-efficiency tradeoff. Although it yields a 50%˘130% precision loss compared to *Conv*, it is significantly faster. For multi-neuron approximations, the table also reports the number of output constraints under the *#Cst.* columns. When the input is 2-D, STM generates more constraints than *Conv*; however, for 3-D and higher-dimensional inputs (the latter not shown in the table), STM produces fewer constraints, which is beneficial for further analysis.

Table 4. Performance results on DNN verification problems with LeakyReLU activations (Time in sec.)

Methods		*DeepPoly*		*PRIMA*			*Polyhedra-Cart*		*Polyhedra-STM*			*Conv*		
Networks	ϵ	*#Ver.*	*Time*	*#Ver.*	*Time*	*#Cst.*	*#Ver.*	*Time*	*#Ver.*	*Time*	*#Cst.*	*#Ver.*	*Time*	*#Cst.*
LeakyReLU 3×100	0.001	21	0.43	+4	1.53	46634	+2	0.89	+5	1.24	4119	+8	4.76	23593
LeakyReLU 6×100	0.0007	28	0.82	+2	3.22	102787	+1	1.08	+1	2.82	7588	+7	7.13	40383
LeakyReLU 9×100	0.0009	30	0.98	+3	3.70	11288	+2	1.14	+3	3.61	7941	+4	9.50	42990

For S-Shaped Functions. Before presenting their results, we briefly describe how support triangles for $\mathcal{S}_{\text{PRIMA}}$ are constructed. Using Fig. 7a as an example, the support triangle ΔFIE can be identified for the upper approximation $\overline{FE}$. For $\overline{EB}$, two scenarios arise: If $\overline{EB}$ is tangent to F, it is used directly as the support line segment. Otherwise, the tangent line at B, which is $\overline{B\tilde{J}}$, forms the support triangle $\Delta E\tilde{J}B$. Unlike non-S-shaped functions, the volume estimation for lifting S-shaped functions can be very imprecise, as single-dimensional polyhedral approximations are typically very thin geometrically. When lifting into $2n$-D space, the resulting volume may be too small to estimate accurately via sampling. For instance, when $n = 3$, often only 0 or 1 point out of $1e8$ samples falling within the polytope. Due to this limitation, we restrict our evaluation to 2-D inputs.

Table 3 summarizes the results. We can see STM consistently outperforms *Cart* in terms of precision. When compared to PRIMA, STM achieves similar precision gains. A notable observation is that $S_{\varkappa}$-*Cart* can be more precise than S_{PRIMA}-*, highlighting the significance of designing tight single-dimensional approximations. In terms of efficiency, STM maintains low runtime while delivering superior precision compared to *Cart* and comparable precision to PRIMA, showcasing its practical applicability. These results emphasize the need for careful design of both single-dimensional approximations and lifting methods to achieve precise analyses. STM demonstrates strong performance in balancing precision and efficiency for S-shaped functions. Moreover, the results show that STM produces more constraints than PRIMA, which is undesirable but appears unavoidable at present. Further optimization is needed to reduce the number of output constraints.

5.3 Deep Neural Network Verification

To evaluate the performance of STM in verifying DNNs, we integrated it into ERAN framework [16], a widely used toolkit developed by ETH researchers for DNN certification. Since ERAN does not support exponential or square functions, our evaluations focused on LeakyReLU functions and S-shaped functions: *Sigmoid* and *Tanh*. We acknowledge that a comprehensive evaluation would ideally involve networks trained on various dataset and with different activation functions, but our goal here is to provide a proof of concept and preliminary results.

For LeakyReLU, we trained three networks with $\alpha = 0.01$ on the CIFAR10 dataset and compared STM against DeepPoly (a leading single-neuron approximation method), PRIMA[5], and *Conv* (the exact convex hull method). For S-shaped functions, we trained six networks on the MNIST dataset, and compared STM with DeepPoly and PRIMA. Additionally, we varied both single-dimensional approximations and lifting techniques for PRIMA and STM to assess the individual contributions of each component. Following PRIMA, we first applied DeepPoly for verification and switched to multi-neuron methods when it failed. Abstraction refinement was disabled for fair comparison.

All methods were evaluated on their ability to verify local robustness under l_∞-norm for the first 100 test samples, with a predefined perturbation radius ϵ for each network. We used three metrics: precision, measured by the number of verified tests; efficiency, measured by the average runtime; and output complexity, measured by the number of constraints in the output approximation.

[5]LeakyReLU is not natively supported in PRIMA; we implemented its handling logic.

Table 5. Performance results on DNN verification problems with S-Shaped activations (Time in sec.)

Methods		*DeepPoly*		S_{PRIMA}-*Cart*		S_{PRIMA}-*STM*			*PRIMA*			$S_{\varkappa}$-*Cart*		$S_{\varkappa}$-*STM*		
Networks	ϵ	*#Ver.*	*Time*	*#Ver.*	*Time*	*#Ver.*	*Time*	*#Cst.*	*#Ver.*	*Time*	*#Cst.*	*#Ver.*	*Time*	*#Ver.*	*Time*	*#Cst.*
Sigmoid 6×100	0.018	31	0.12	+2	1.07	+5	4.09	98095	+7	9.77	248528	+4	0.99	+8	3.44	63489
Sigmoid 6×200	0.012	40	0.54	+0	1.40	+1	4.31	127182	+1	14.96	420592	+2	1.39	+4	4.11	103191
Sigmoid 9×100	0.017	37	0.23	+1	1.31	+2	4.89	148891	+2	21.20	359039	+1	1.2	+1	4.16	92085
Tanh 6×100	0.005	66	0.12	+6	0.55	+13	2.09	76819	+16	11.13	232867	+8	0.54	+15	1.90	59698
Tanh 6×200	0.004	51	0.50	+4	1.83	+8	5.90	120814	+12	18.23	375740	+6	1.80	+12	6.08	97582
Tanh 9×100	0.003	49	0.21	+1	1.20	+3	4.31	120291	+4	21.64	347870	+3	1.09	+5	4.32	91081

Table 4 reports the results for DNNs with LeakyReLU activations. From the table, we can see that *Conv* achieves the highest precision but is the slowest, while DeepPoly is the fastest but least precise. STM and *Polyhedra*–STM achieve comparable precision, with the latter being faster. When using the same single-neuron approximations, STM consistently achieves higher precision than *Cart*. In terms of output complexity, STM produces around 30% fewer constraints than PRIMA.

For DNNs with S-shaped activation functions, the results are summarized in Table 5. To facilitate deeper understanding, we analyze the outcomes from three perspectives:

Single-Neuron Approximations. First, we examine the performance of single-neuron approximation methods, including DeepPoly, S_{PRIMA}-*Cart* and $S_{\varkappa}$-*Cart*. The results show that S_{*}-*Cart* generally achieves higher precision but than takes more time than DeepPoly. This is because S_{*} can take up to two linear constraints for each upper or lower approximation, while DeepPoly employs only one constraint per bound, simplify the approximation but sacrificing precision. When comparing $S_{\varkappa}$-*Cart* to S_{PRIMA}-*Cart*, $S_{\varkappa}$ proves to be a better single-neuron approximation as it achieves higher precision while maintaining comparable efficiency.

Lifting Methods. Next, we assess the effectiveness of STM as a lifting technique by comparing its performance with *Cart* when applied to the same single-neuron approximations. (S_{PRIMA}-*Cart* v.s. S_{PRIMA}-*STM*, $S_{\varkappa}$-*Cart* v.s. $S_{\varkappa}$-*STM*). The results reveal that STM consistently verifies more tests than *Cart*, demonstrating that STM significantly enhances precision in neural network verification. Although STM requires more time per analysis, its absolute runtime remains low, ensuring that the increased precision does not come at high computational cost.

Multi-Neuron Approximations. Finally, we compare STM with PRIMA. The results indicate that STM and PRIMA verify a similar number of test cases, showing that they achieve comparable precision, while STM completes the analysis in much less time. This efficiency gain, combined with the relative simplicity of implementing STM, makes it a promising advancement for neural network verification. These findings suggest that STM offers a balanced approach, achieving high precision while remaining computationally efficient.

Summary. While STM does not outperform existing methods in every aspect, our preliminary results indicate that it strikes a good balance between lifting precision and computational cost. We therefore consider STM a viable candidate and a promising direction for future research. In particular, it not only offers a new perspective on constructing multi-neuron approximations, i.e. by lifting single-neuron approximations, but also provides high applicability and low development overhead — both of which are critical when exploring new activation functions.

6 Related Works

This part surveys related works in convex lifting, convex hull approximation, and DNN verification.

6.1 Convex Lifting

Convex lifting [2, 30] is a mathematical technique used to project lower-dimensional problems into higher-dimensional spaces, making them more tractable or offering new insights. It has been applied in various fields such as optimization, and computational geometry. In combinatorial optimization, Lift-and-Project Methods [3] are used to strengthen linear programming (LP) relaxations of integer programming problems. By introducing additional variables and constraints, these methods yield a closer approximation to the convex hull of feasible solutions, thereby enhancing solution quality. In computational geometry, [2] takes convex lifting to compute the convex hull of points in $\mathbb{R}^n$ by mapping them into $\mathbb{R}^{n+1}$ and using algorithms like Quickhull in higher-dimensional space. This simplifies the computation by leveraging the geometric properties of convex sets in higher dimensions. Our work on convex approximation lifting is conceptually related and the idea behind our proposed approach may have the potential to be integrated into existing convex lifting methods.

6.2 Convex Hull Approximation

In computational geometry, computing convex hulls is a foundational task essential to understanding the structure of multidimensional data. Classical algorithms such as Quickhull [4], Graham's scan [19], and gift wrapping [9] provide efficient methods for $2D$ and $3D$, with recent work extending these methods to higher dimensions [10, 22]. Convex hull approximation, which seeks near-optimal hulls at reduced computational cost, employs strategies like boundary sampling and core-set construction to balance precision with efficiency [5, 21, 25]. From this perspective, our work contributes a specialized family of convex approximation problems that involve lower-dimensional approximations and function lifting. The approach proposed in this work may have the potential to optimize existing convex hull approximation methods.

6.3 DNN Verification

The formal verification of DNNs is essential to ensuring their reliability in safety-critical applications. Verification methods [1, 6–8, 11, 15, 20, 23, 24, 26, 40, 44] are typically divided into two categories: exact and approximate. Exact methods [1, 7, 8, 11, 23, 24, 44] rely on exhaustive search techniques and satisfiability solvers to explore all possible network behaviors. While precise, these methods often suffer from scalability limitations. In contrast, approximate methods, which are the focus of this work, trade completeness for scalability. They typically construct over-approximations of neural networks using interval abstraction [18], linear constraints [15], zonotope abstraction [34], fragments of polyhedra [35, 41, 45], or duality frameworks [12, 13, 42, 43]. WraLU [27] and PRIMA [29] represent two recent advances that focus on multi-neuron approximations for activations. Our approach tackles the problem from a new angle, that is to construct multi-neuron approximations by systematically leveraging single-neuron ones, which is very different from existing methods.

7 Conclusion

This work bridges the gap between single-neuron and multi-neuron approximation methods in DNN verification through approximation lifting. In particular, we formulate the lifting process as a novel CAL problem and propose a constructive approach, STM, to solving it. We also theoretically establish the correctness of STM and empirically evaluate its performance.

Looking ahead, we expect our work to inspire further research in fields beyond DNN verification and to encourage more researchers to contribute to the verification community. Indeed, the H-P pairing technique for locating high-dimensional hyperplanes and the decomposition mechanisms for managing function approximations appear to be broadly applicable, suggesting that STM might have more applications beyond DNN verification.

Acknowledgments

We thank the anonymous reviewers for their critical comments and constructive suggestions. This paper represents the third work in our research series on the formal verification of DNNs. We would like to thank Jun Sun for his valuable support on the first work, SURGEON [26], and Guangdong Bai for his contribution to the second work, WraLU [27]. We also thank Zhendong Su for his lasting influence on our research taste. The views expressed in this work does not necessarily reflect the position or the policy of any government and no official endorsement should be inferred.

References

[1] 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)*, Andrea Lodi and Viswanath Nagarajan (Eds.). Springer, 27–42. doi:10.1007/978-3-030-17953-3_3

[2] David Avis and Komei Fukuda. 1991. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. In *Proceedings of the seventh annual symposium on Computational geometry*. 98–104. doi:10.1145/109648.109659

[3] Egon Balas, Sebastián Ceria, and Gérard Cornuéjols. 1993. A lift-and-project cutting plane algorithm for mixed 0–1 programs. *Mathematical programming* 58, 1-3 (1993), 295–324. doi:10.1007/BF01581273

[4] 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. doi:10.1145/235815.235821

[5] Jon Louis Bentley, Franco P Preparata, and Mark G Faust. 1982. Approximation algorithms for convex hulls. *Commun. ACM* 25, 1 (1982), 64–68. doi:10.1145/358315.358392

[6] 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. doi:10.1609/aaai.v34i04.5729

[7] 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).

[8] 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*, Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (Eds.). 4795–4804.

[9] Donald R. Chand and Sham S. Kapur. 1970. An Algorithm for Convex Polytopes. *J. ACM* 17, 1 (1970), 78–86. doi:10.1145/321556.321564

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

[11] 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.

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

[13] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A. Mann, and Pushmeet Kohli. 2018. A Dual Approach to Scalable Verification of Deep Networks.. In *UAI*, Amir Globerson and Ricardo Silva (Eds.). AUAI Press, 550–559.

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

[15] Rüdiger Ehlers. 2017. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks.. In *ATVA (Lecture Notes in Computer Science, Vol. 10482)*, Deepak D'Souza and K. Narayan Kumar (Eds.). Springer, 269–286. doi:10.1007/978-3-319-68167-2_19

[16] ETH SRI Team. 2022. ERAN: ETH Robustness Analyzer for Neural Networks. https://github.com/eth-sri/eran

[17] 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. doi:10.1109/SP.2018.00058

[18] 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. doi:10.1109/ICCV.2019.00494

[19] Ronald L. Graham. 1972. An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. *Inf. Process. Lett.* 1 (1972), 132–133. https://api.semanticscholar.org/CorpusID:45778703

[20] 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)*, Rupak Majumdar and Viktor Kuncak (Eds.). Springer, 3–29. doi:10.1007/978-3-319-63387-9_1

[21] S-O Jeong and BU Park. 2006. Large sample approximation of the distribution for convex-hull estimators of boundaries. *Scandinavian Journal of Statistics* 33, 1 (2006), 139–151. doi:10.1111/j.1467-9469.2006.00452.x

[22] Michael Joswig. 2003. Beneath-and-Beyond Revisited. In *Algebra, Geometry, and Software Systems*, Michael Joswig and Nobuki Takayama (Eds.). Springer, 1–21.

[23] 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)*, Rupak Majumdar and Viktor Kuncak (Eds.). Springer, 97–117. doi:10.1007/978-3-319-63387-9_5

[24] 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)*, Isil Dillig and Serdar Tasiran (Eds.). Springer, 443–452. doi:10.1007/978-3-030-25540-4_26

[25] Georgiy Klimenko and Benjamin Raichel. 2021. Fast and exact convex hull simplification. *arXiv preprint arXiv:2110.00671* (2021).

[26] Jiaying Li, Guangdong Bai, Long H Pham, and Jun Sun. 2023. Towards an Effective and Interpretable Refinement Approach for DNN Verification. In *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS)*. IEEE, 569–580. doi:10.1109/QRS60937.2023.00062

[27] Zhongui Ma, Jiaying Li, and Guangdong Bai. 2024. ReLU Hull Approximation. *Proceedings of the ACM on Programming Languages* 8, POPL (2024), 2260–2287. doi:10.1145/3632917

[28] Samvid Mistry, Indranil Saha, and Swarnendu Biswas. 2022. An MILP Encoding for Efficient Verification of Quantized Deep Neural Networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 11 (2022), 4445–4456. doi:10.1109/TCAD.2022.3197697

[29] 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), 1–33. doi:10.1145/3498704

[30] Ngoc Anh Nguyen, Martin Gulan, Sorin Olaru, and Pedro Rodriguez-Ayerbe. 2017. Convex lifting: Theory and control applications. *IEEE Trans. Automat. Control* 63, 5 (2017), 1243–1258. doi:10.1109/TAC.2017.2737234

[31] Long H Pham and Jun Sun. 2022. Verifying neural networks against backdoor attacks. In *International Conference on Computer Aided Verification*. Springer, 171–192.

[32] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. 2018. Semidefinite relaxations for certifying robustness to adversarial examples.. In *NeurIPS*, Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (Eds.). 10900–10910.

[33] Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin T. Vechev. 2019. Beyond the Single Neuron Convex Barrier for Neural Network Certification.. In *NeurIPS*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché Buc, Emily B. Fox, and Roman Garnett (Eds.). 15072–15083.

[34] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T. Vechev. 2018. Fast and Effective Robustness Certification.. In *NeurIPS*, Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (Eds.). 10825–10836.

[35] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. 2019. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.* 3, POPL (2019), 41:1–41:30. doi:10.1145/3290354

[36] 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)*, Yoshua Bengio and Yann LeCun (Eds.).

[37] Vincent Tjeng, Kai Xiao, and Russ Tedrake. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. arXiv:1711.07356 [cs.LG]

[38] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. 2019. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE symposium on security and privacy (SP)*. IEEE, 707–723. doi:10.1109/SP.2019.00031

[39] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium (USENIX Security 18)*. 1599–1614.

[40] 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.

[41] 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)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, 5273–5282.

[42] 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)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, 5283–5292.

[43] Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J Zico Kolter. 2018. Scaling provable adversarial defenses. *Advances in Neural Information Processing Systems* 31 (2018).
[44] 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)*.
[45] 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*, Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (Eds.). 4944–4953.

Received 2024-11-14; accepted 2025-03-06