Synthesizing Boxes Preconditions for Deep Neural Networks

Zengyu Liu
State Key Laboratory of Complex & Critical Software Environment, College of Computer Science and Technology, National University of Defense Technology
Changsha, China
liuzengyu21@nudt.edu.cn

Liqian Chen
State Key Laboratory of Complex & Critical Software Environment, College of Computer Science and Technology, National University of Defense Technology
Changsha, China
lqchen@nudt.edu.cn

Wanwei Liu
State Key Laboratory of Complex & Critical Software Environment, College of Computer Science and Technology, National University of Defense Technology
Changsha, China
wwliu@nudt.edu.cn

Ji Wang*
State Key Laboratory of Complex & Critical Software Environment, College of Computer Science and Technology, National University of Defense Technology
Changsha, China
wj@nudt.edu.cn

Abstract

Deep neural network (DNN) has been increasingly deployed as a key component in safety-critical systems. However, the credibility of DNN components is uncertain due to the absence of formal specifications for their data preconditions, which are essential for ensuring trustworthy postconditions. In this paper, we propose a guess-and-check-based framework *PreBoxes* to automatically synthesize Boxes sufficient preconditions for DNN concerning rich safety and robustness postconditions. The framework operates in two phases: the *guess* phase generates potentially complex candidate preconditions through heuristic methods, while the *check* phase verifies these candidates with formal guarantees. The entire framework supports automatic and adaptive iterative running to obtain weaker preconditions as well. Such resulting preconditions can be leveraged to shield DNN for safety and enhance the interpretability of DNN in application. *PreBoxes* has been evaluated on over 20 models with 23 trustworthy properties of 4 benchmarks and compared with 3 existing typical schemes. The results show that not only does *PreBoxes* generally infer weaker non-trivial sufficient preconditions for DNN than others, but also it expands competitive capabilities to handle both *complex properties* and *Non-ReLU complex structured networks*.

CCS Concepts

• **Software and its engineering** → **Automated static analysis**; **Correctness**; **Dynamic analysis**; • **Computing methodologies** → *Machine learning*.

Keywords

Precondition Synthesis, Neural Network, Boxes, Robustness, Safety

*Corresponding author.

ACM Reference Format:
Zengyu Liu, Liqian Chen, Wanwei Liu, and Ji Wang. 2024. Synthesizing Boxes Preconditions for Deep Neural Networks. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24), September 16–20, 2024, Vienna, Austria.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3650212.3680393

1 Introduction

Deep learning has been widely used in safety- or mission-critical applications designed to function in open environments for perception and decision-making [7, 34], whose minor errors might cause irreversible damage. Despite the outstanding empirical achievements of deep neural networks (DNNs), formally guaranteeing their specifications is still a significant and challenging task partly due to the data-driven implicit black box of DNNs themselves and the uncertainty of complex environments outside when combined into autonomous systems.

Towards guaranteeing desired specifications (also referred to as contracts), formal verification techniques for DNNs early days [13, 30–32, 35] pay particular attention to inferring postcondition, that is, to provide a qualitative analysis of whether a certain property holds globally for the given small input domain (such as a ϵ-perturbation norm neighborhood). However, such a perspective has difficulty in offering an indirect quantitative analysis on the specification when the expected property is not globally satisfied on the whole input domain, that is, to solve which concrete partial input space still meets the desired property. Such attention on precondition also provides valuable specification descriptions of DNNs, helping improve their own interpretability. What's more, such precondition synthesis helps provide a reliability guarantee for DNNs when applied at the system level. Before the deployment of autonomous systems, precondition synthesis techniques could be used to make quality assessments on multiple similar trained DNNs, so as to compare and select appropriate ones matching the design specifications of system components containing DNNs [26], which furthermore helps achieve a safe and controllable system integration. At the running time, the synthesized preconditions could play a role in runtime monitoring for real-time shielding illegal inputs from open environments for DNN components and making timely adjustments for other system components' task allocation, such as by using the Simplex architecture [27].

Recently, the synthesis of DNN preconditions has been gradually attracting attention. Although the over-approximation of DNN preconditions [17] contains all legal inputs, it also introduces illegal inputs that violate the desired property. Such approximation makes it impossible to make indirect quantitative analysis on DNNs themselves and select appropriate DNNs as system components for desired specifications. The exact synthesis of DNN preconditions is the optimal solution to the research problem ideally, but it is intractable due to the complexity increasing exponentially with the scale of DNNs. To the best of our knowledge, current existing work [8, 21, 24] typically supports solving for several variable input features and is only limited to small ReLU DNNs. Therefore, it is valuable to under-approximate DNN preconditions. As far as we know, the types of DNNs that most current under-approximation work supports are very limited [10, 12, 36]. Most of these works only design for or experiment on ReLU DNNs, and few mention piecewise linear DNNs. Furthermore, the desired properties directly supported by existing works are also relatively simple and usually represent a convex output set as conjunctions of linear constraints. Such syntax usually expresses similar semantic covering local robustness properties and only part of safety properties [10, 12, 19, 36].

We propose a guess-and-check-based framework *PreBoxes* in a gray-box way, which automatically synthesizes under-approximate Boxes preconditions as weak as possible with no special limitation on DNN types and supporting for rich desired properties. Considering the well-expression and easy-readability of the high-level specification intuitively, the *guess* phase uses Boxes as the template to generate non-trivial precondition candidates. Based on our insight into the spatial structure of DNN decision boundary in the input domain, we design a cost-effective directional boundary sampling heuristic to quickly predict the split sub-regions of the input domain that have more potential to be preconditions themselves. We further design an aggregation heuristic for all the potential split sub-regions based on their spatial location information concerning each other, so as to obtain the non-trivial Boxes candidates of preconditions and reduce the overhead of verification in the *check* phase. The heuristics here are in black-box style relative to DNN itself. The *check* phase then verifies whether the required specification is formally guaranteed, i.e. checking which of these candidates are true preconditions by invoking the neural network verifier. Furthermore, the guess-and-check framework could be performed iteratively to weaken the precondition solutions as close to the exact preconditions as possible. To summarize, our contributions are as follows.

- We propose a guess-and-check-based framework in a gray-box way to under-approximately synthesize weak Boxes preconditions with no special limitation on DNN types and supporting for rich desired properties.
- We design a cheap directional boundary sampling heuristic to efficiently predict which potential parts of the input domain are preconditions themselves as well as a spatial aggregation algorithm so as to obtain the non-trivial Boxes candidates and further improve verification efficiency.
- We have implemented the framework as a tool called *PreBoxes*. We conducted experiments over 20 models with 23 properties of 4 benchmarks, *Syn*[19], *CartPole*, *ACASXu* and *NN4Sys* in VNN-COMP[22], and compared *PreBoxes* with 3 existing schemes, *Uniform*, *NonUniform* in [19] and *PreimageAppro* in [36]. The results show *PreBoxes* can generally infer weaker non-trivial sufficient preconditions for DNN than others and has competitive capabilities to deal with *complex properties* and *Non-ReLU complex structured networks*.

As far as we know, although *ACASXu* has been discussed in previous work, we are the first to infer preconditions on *ACASXu* without predetermined input feature values and with less type-restricted and richer properties. Additionally, we have extended DNN precondition synthesis beyond piecewise linear DNNs like *NN4Sys*, which has never been done before.

2 Problem Formulation

Given the desired property $\mathcal{P}$ as the postcondition for a given DNN $f : \mathbb{R}^n \to \mathbb{R}^m$, the precondition of f can be treated as a set of all inputs $x \in \mathbb{R}^n$ that are mapped to an element of $\mathcal{P}$ by f. The under-approximation of precondition could be treated as any subset of the exact one, which is also referred to as sufficient precondition. We hope to infer non-trivial sufficient precondition as weak as possible, i.e., cover as much of the input space as possible. As usual, a precondition can be expressed as a predicate or a set characterized by the predicate. We don't make a strict distinction here.

We define a set $\mathcal{B} \subseteq \mathbb{R}^n$ as a n-dimensional *Box* iff it is expressible as $\mathcal{B} = \left\{x \in \mathbb{R}^n \mid \bigwedge_{i=1}^{n}(x^i \in [x_l^i, x_u^i]), where\ x_l^i, x_u^i \in \mathbb{R}\right\}$. The set of all boxes of $\mathbb{R}^n$ is denoted as $\mathbb{B}^n$. A set $\mathcal{BS} \subseteq \mathbb{R}^n$ conforms to the *Boxes* iff there exist n-dimensional boxes $\mathcal{B}_1, \cdots, \mathcal{B}_k$ such that $\mathcal{BS} = \bigcup_{i=1}^{k} \mathcal{B}_i$. The set of all sets of boxes of $\mathbb{R}^n$ is denoted as $\mathbb{BS}^n$ [11]. The input features of DNN usually have physical meaning in applications, so we consider the features to have limited numerical ranges, i.e. the input domain of DNN is bounded. We also require sufficient precondition solutions of DNN in the form of *Boxes*. Additionally, the postcondition $\mathcal{P}$ can be expressed as the formulas over linear inequalities supporting for mixed conjunctive and disjunctive form, which include many properties like local robustness and complex safety properties.

In summary, the synthesis problem here could be formulated as follows: Given a DNN f under a bounded input domain $\mathcal{I} \in \mathbb{B}^n$ and the desired property $\mathcal{P}$ expressed as mixed conjunctive and disjunctive form of boolean expressions over linear inequalities, how to calculate the *Boxes* sufficient precondition $f_{\mathcal{I}}^{-1}(\mathcal{P})$ as weak as possible.

$$f_{\mathcal{I}}^{-1}(\mathcal{P}) := \{\mathcal{BS} \in \mathbb{BS}^n \mid \mathcal{BS} \subseteq \mathcal{I} \wedge f(x) \in \mathcal{P}, where\ \forall x.x \in \mathcal{BS}\} \quad (1)$$

3 Methodology

3.1 Overview

Our solution provides a method for automatically synthesizing sufficient preconditions for DNNs in a guess-and-check framework, as depicted in Figure 1. At the *guess* phase, we reconstruct the original DNN to a featured DNN with a unified and standardized expression meaning of its output based on the desired property. At the same time, we adopt the idea of partitioning first and merging later on the input domain to obtain non-trivial boxes sufficient precondition candidates through cost-effective heuristics. Specifically, the input domain is split into multiple finer boxes at first. Then we design a heuristic directionally sampling on the boundaries of each partitioned space to efficiently select those sub-boxes having greater potential as preconditions themselves. Furthermore, to reduce the overhead of redundant verification and increase the readability of final solutions, we design an aggregation heuristic for merging small candidates into larger and fewer boxes based on spatial structure rather than obtaining a great many trivial candidates. The subsequent *check* phase is to perform provable verification on the candidate set by invoking the DNN verifier so as to generate truly sufficient *Boxes* preconditions. After such a single guess-and-check process, the solutions have already been provable under-approximation of the exact precondition. Furthermore, to make sufficient preconditions weaker and get to the exact precondition as close as possible, our framework also supports performing adaptive iterative running on the candidates for which the verifier fails to determine a clear result or for which it is deemed unsafe, thereby solving weaker preconditions, i.e. covering input domains as much as possible.

To dive into detail in the following subsections, we first elaborate on how to generate non-trivial sufficient preconditions, and then provide an adaptive iterative version helping obtain weaker provable sufficient preconditions.

3.2 Infer Non-Trivial Sufficient Precondition

3.2.1 Building Featured DNN. Inspired by DNN verification [6], we want to bring the benefits of unified canonical representation to the DNN precondition synthesis problem here as well.

Let $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ be the original DNN and $\mathcal{P}$ be the desired property expressed in the *mixed conjunctive and disjunctive form* of boolean expression over linear constraints. What we integrate into the featured DNN $\bar{f} : \mathbb{R}^n \rightarrow \mathbb{R}^1$ is the violation of $\mathcal{P}$ (referred as $\neg\mathcal{P}$) as addition layers $f_{add} : \mathbb{R}^m \rightarrow \mathbb{R}^1$ after f. Suppose the output of f is $y \in \mathbb{R}^m$. No matter $\neg\mathcal{P}$ is single non-relational linear inequality (like $a \times y_i \leq b, where\ a, b \in \mathbb{R},\ i \in \{1, ..., m\}$) or single relational linear inequality (like $y_i \leq y_j, where\ i \neq j$, $i, j \in \{1, ..., m\}$), they can be regularized into vector multiplication form $C^T y + d \leq 0,\ where\ C \in \mathbb{R}^m, d \in \mathbb{R}$ and treated as a fully connected layer $f_{add} = C^T y + d$. Besides, conjunction or disjunction could be handled by *maxpooling* unit. If $\neg\mathcal{P}$ is $\bigwedge_1^k \left(y_i \leq y_j\right)_r$, $where\ k \in \mathbb{Z}, i \neq j, i, j \in \{1, \cdots, m\}, r \in \{1, \cdots, k\}$, we make f_{add} as $max(\left(y_i - y_j\right)_r), where\ r\ \in \{1, \cdots, k\}$. If $\neg\mathcal{P}$ is $\bigvee_1^k \left(y_i \leq y_j\right)_r$, considering $min(\left(y_i \leq y_j\right)_r) \Leftrightarrow -max(-\left(y_i \leq y_j\right)_r)$, we set f_{add} as $-max(-\left(y_i - y_j\right)_r), where\ r\ \in \{1, \cdots, k\}$. The other forms can be handled with a combination of the above rules.

Such a normalization makes the output of featured DNN $\bar{f}$ have a unified meaning. The output of a positive scalar indicates the input satisfies $\mathcal{P}$. A negative scalar output implies the input satisfies $\neg\mathcal{P}$. And a zero scalar output indicates that input falls on the *decision boundary* $\left\{x \in \mathbb{R}^n | \bar{f}(x) = 0\right\}$. With $\bar{f}$, we could reformulate the *Boxes* sufficient precondition in a unified canonical representation:

$$f_{\mathcal{I}}^{-1}(\mathcal{P}) := \left\{\mathcal{BS} \in \mathbb{BS}^n | \mathcal{BS} \subseteq \mathcal{I} \wedge \bar{f}(x) > 0,\ where\ \forall x . x \in \mathcal{BS}\right\} \quad (2)$$

Such a canonical representation would reduce the cost of processing complex properties, e.g. containing several OR clauses, using a single procedure rather than dismantling enumerator subs [13]. Besides, the featured DNN helps convert the satisfiability problem to a simpler numerical positivity judgment. Furthermore, this form facilitates our understanding of the relationship between *decision boundary* and preconditions, as discussed in Section 3.2.3 later.

3.2.2 Input Splitting. When there is no prompt about the true preconditions, it is unwise to hand over the entire domain to the verifier because as long as there is a counterexample (i.e., an input that violates the expected property $\mathcal{P}$), the verifier would declare the entire domain as unsafe based on a qualitative judgment, even if there are still stronger preconditions inside the domain. To gain more insight into true preconditions, we divide the entire domain into a set of aligned boxes covering the original.

We instantiate two schemes for splitting the input domain $\mathcal{I} \in \mathbb{B}^n$, namely *dimension splitting* and *counterexample splitting*. Figure 2 illustrates two-dimensional examples. When without any hint of $\mathcal{I}$, the *dimension splitting* provides a feasible approach to obtain fine boxes by dividing evenly within each dimension range. It allows for the configuration of distinct segment times across various dimensions to accommodate differing feature ranges, as shown in Figure 2(a). Additionally, the findings from DNN verification [22] suggest that a counterexample is often situated in close spatial proximity to other counterexamples. This characteristic aids in gathering and processing many possible counterexamples during input domain splitting, helping to get more potential candidates later. The *counterexample splitting* determines a definite violation region Ω around c_x as a l_∞-norm ball around c_x with a radius of ϵ, that is $\Omega = \{x \in \mathbb{R}^n | \ \|x - c_x\|_\infty \leq \epsilon\}$, and then use the lower and upper boundaries of Ω in each dimension as breakpoints, e.g. $\left[x_i^{splitl}, x_i^{splitu}\right]$, where $i \in \{1, \cdots, n\}$ shown in Figure 2(b), to partition a set of aligned sub-boxes. We detail the adaptive selection of input splitting strategies in Section 3.3.

3.2.3 State Prediction. The unified output meaning of the featured DNN $\bar{f}$ facilitates to reveal of the relationship among *decision boundary*, precondition, and violation space. The featured DNN $\bar{f}$ could be viewed as a continuous function on a compact set based on the universal approximation theorem of DNN [3], so there must be a zero value between positive and negative values. The *decision boundary* $\left\{x \in \mathbb{R}^n | \bar{f}(x) = 0\right\}$ reflects the critical situation of whether the property is satisfied or not, and the precondition $\left\{x \in \mathbb{R}^n | \bar{f}(x) > 0\right\}$ and violation space $\left\{x \in \mathbb{R}^n | \bar{f}(x) < 0\right\}$ are on both sides of it, as the blue line and indicative arrows on both sides shown in Figure3(a). In this 2D example, the yellow arrow side is the weakest target of synthesis.

We hope to obtain more hints about preconditions by positioning *decision boundary* among partitioned boxes. We evenly sample along the boundaries of each box directionally, evaluating the positivity of all sampled data using the featured DNN $\bar{f}$. This process allows us to identify definite violation space and continue discussing potential boxes as candidates for subsequent analysis. We illustrate four sub-boxes $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D} \in \mathbb{B}^2$ in Figure 3(a). $\mathcal{B}$ and $\mathcal{C}$ represent scenarios where the *decision boundary* intersects the box when both positive and negative sample points are present

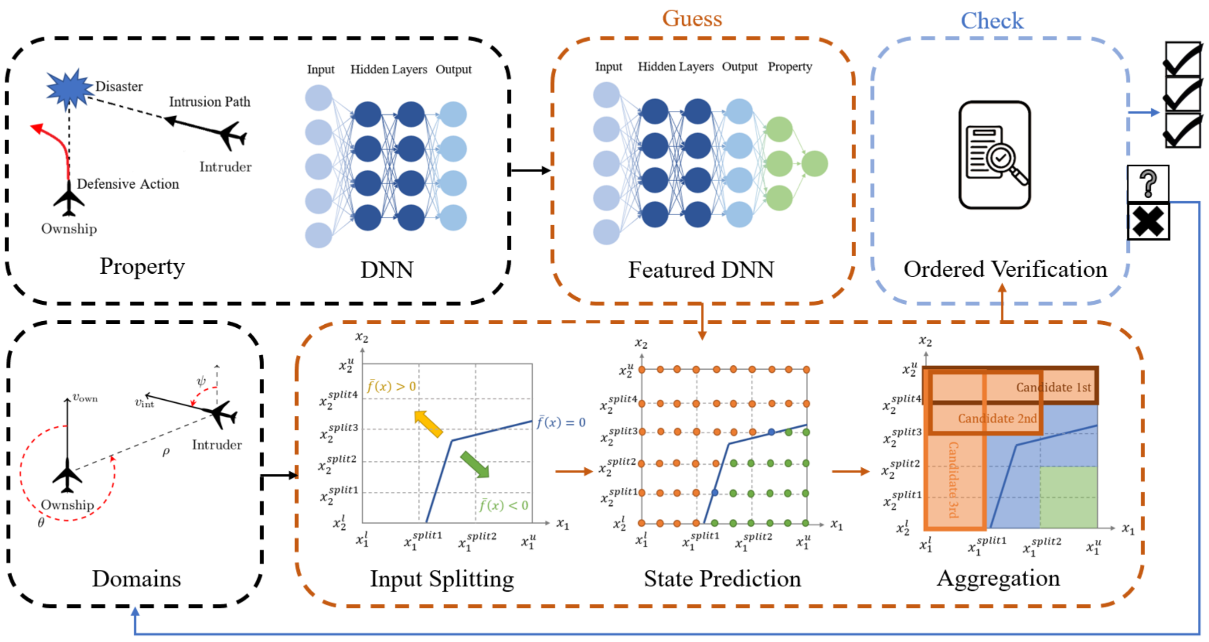


Figure 1: Overview of our *PreBoxes*

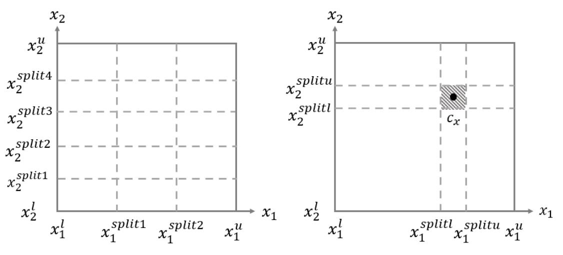


Figure 2: Demonstration of Two Input Splitting Schemes.

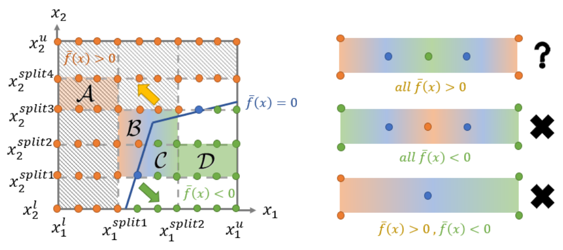


Figure 3: Demonstration of State Prediction.

simultaneously, without any mandatory zero points. These scenarios constitute violation space as a whole in the presence of counterexamples. All positive sample points, like $\mathcal{A}$, suggest potential precondition candidates, while negative points, such as $\mathcal{D}$, indicate violation boxes. Ultimately, all dotted sub-boxes in the figure represent potential precondition candidates. For a space $\Omega \subseteq \mathbb{R}^n$, we use $predict(\Omega) = potential$ to denote its potential as a precondition according to our method, and $predict(\Omega) = violated$ indicates $\exists x. x \in \Omega, \bar{f}(x) < 0$.

Although $\bar{f}$ can be viewed as a continuous function, it is not necessarily monotonic. As shown in Figure 3 (b), when all the boundary sampling points satisfy property $\mathcal{P}$, the entire box may still cross the *decision boundary*. As long as there is a counterexample in the box, it as a whole input domain does not satisfy $\mathcal{P}$.

Proposition 3.1. *Our state prediction method with the above directional boundary sampling is sound for predicting violation space.*

3.2.4 Aggregation. A Boxes $\mathcal{BS}_{ori} \in \mathbb{BS}^n$ composed by k potential sub-boxes $\mathcal{B}_{ori_1},...,\mathcal{B}_{ori_k}$ can be obtained after state prediction. It is trivial to send all such sub-boxes to the verifier to get a provable guarantee. However, the frequent invocation of the verifier would incur significant verification overhead. Consequently, we will aggregate the scattered boxes in $\mathcal{BS}_{ori}$ into larger boxes cost-effectively in order to reduce the number of verifier invocations. Note that, we limit the aggregated sub-region to *Box* and the same total space covered by *Boxes* for keeping the readability of DNN contract. The goal is formalized as below: Given $\mathcal{BS}_{ori}$ with k boxes, the problem is to solve a *Boxes* $\mathcal{BS}_{agg} \in \mathbb{BS}^n$ covered by j potential boxes $\mathcal{B}_{agg_1},...,\mathcal{B}_{agg_j}$, such that $\mathcal{BS}_{agg} = \mathcal{BS}_{ori}$ and $j \leq k$, i.e. $\forall \mathcal{B}_{ori_q}. \exists \mathcal{B}_{agg_p}. (\mathcal{B}_{ori_q} \subseteq \mathcal{B}_{agg_p}) \wedge (predict(\mathcal{B}_{agg_p}) = potential)$, where $q \in \{1, ..., k\}$, $p \in \{1, ..., j\}$. Subscripts are omitted by default to indicate objects in common situations like $\mathcal{B}_{agg}$ and $\mathcal{B}_{ori}$.

It would be ideal to construct the larger potential box $\mathcal{B}_{agg}$ as the *maximal expansion* of $\mathcal{B}_{ori}$ intuitively. That is if there exists another box $\mathcal{B}$ such that $\mathcal{B}agg \subseteq \mathcal{B}$ and $predict(\mathcal{B}) = potential$, then $\mathcal{B} = \mathcal{B}agg$. However, it has to enumerate every larger box containing $\mathcal{B}_{ori}$ and check whether it is valid *maximal expansion* by using brute-force search. The sheer size of such search space, compounded by the exponential increase in dimensionality, renders brute-force search impractical, even for low-dimensional $\mathcal{B}_{ori}$.

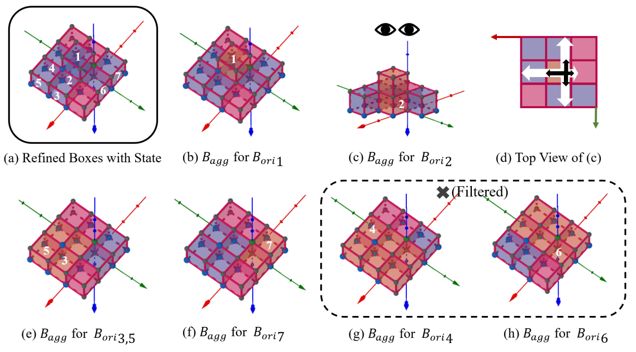
Algorithm 1 Aggregation

Input: the refined boxes obtained by input splitting $\mathbb{I}$ and the according prediction states of refined boxes $\mathbb{S}$

Output: candidates $\mathbb{C}$ waiting for verification

```
1:  $\mathbb{C} \leftarrow \{\}$ ▷ $\mathcal{BS}_{agg} = \bigcup_{i=1}^{j} \mathcal{B}_{agg_i}, where\ \mathcal{B}_{agg_i} \in \mathbb{C}, i \in \{1, ..., j\}$
2:  $\mathbb{B}_{ori}, \mathbb{B}_{vio} \leftarrow$ classify($\mathbb{I}, \mathbb{S}$)
3:  $d \leftarrow$ pickMainSequence($\{1, ..., n\}$)
4:  $\mathbb{B}_{ori} \leftarrow$ ascendingSort($\mathbb{B}_{ori}, d$)
5:  for each $\mathcal{B}_{ori} = \left\{x \in \mathbb{R}^n \mid \bigwedge_{i=1}^{n} (x^i \in [x_l^i, x_u^i])\right\}$ in $\mathbb{B}_{ori}$ do
6:      $\mathcal{B}_{A^2} \leftarrow$ findingAboveAdjacentORI($\mathcal{B}_{ori}, d, \mathbb{I}, \mathbb{S}$)
        ▷ The value decreasing direction on $d$ is designated as above.
7:      if $predict(\mathcal{B}_{A^2}) = potential$ then
8:          $\mathcal{B}_{ori}.height \leftarrow \mathcal{B}_{A^2}.height + (\mathcal{B}_{ori}.x_u^d - \mathcal{B}_{ori}.x_l^d)$
9:      else
10:         $\mathcal{B}_{ori}.height \leftarrow \mathcal{B}_{ori}.x_u^d - \mathcal{B}_{ori}.x_l^d$
11:     end if
12:     for each $i$ in $\{1, ..., n\} / \{d\}$ do
13:         $\mathcal{B}_{ori}.i.leftTemp \leftarrow$ traverse($\mathcal{B}_{ori}, i, \mathbb{I}, \mathbb{S}, left$)
14:         $\mathcal{B}_{ori}.i.rightTemp \leftarrow$ traverse($\mathcal{B}_{ori}, i, \mathbb{I}, \mathbb{S}, right$)
            ▷ The value decreasing direction on $i$ is designated as left
            while the opposite is right.
15:         if $predict(\mathcal{B}_{A^2}) = potential$ then
16:             $\mathcal{B}_{ori}.i.left \leftarrow$ max($\mathcal{B}_{ori}.i.leftTemp, \mathcal{B}_{A^2}.i.left$)
17:             $\mathcal{B}_{ori}.i.right \leftarrow$ min($\mathcal{B}_{ori}.i.rightTemp, \mathcal{B}_{A^2}.i.right$)
18:         else
19:             $\mathcal{B}_{ori}.i.left \leftarrow \mathcal{B}_{ori}.i.leftTemp$
20:             $\mathcal{B}_{ori}.i.right \leftarrow \mathcal{B}_{ori}.i.rightTemp$
21:         end if
22:     end for
23:     $\mathcal{B}_{agg} \leftarrow$ span($\mathcal{B}_{ori}.height, (\mathcal{B}_{ori}.i.left, \mathcal{B}_{ori}.i.right), i \in \{1, ..., n\} / \{d\}$)
24:     if $\mathcal{B}_{agg} \subseteq \mathcal{BS}_{ori}$ then
25:         $\mathbb{C} \leftarrow \mathbb{C} \cup \{\mathcal{B}_{agg}\}$ ▷ $\mathcal{BS}_{ori} = \bigcup_{i=1}^{k} \mathcal{B}_{ori_i}, where\ \mathcal{B}_{ori_i} \in \mathbb{B}_{ori}, i \in \{1, ..., k\}$
26:     end if
27: end for
```

To alleviate such a dilemma, we design a cost-effective practical algorithm (Algorithm 1) to instantiate $\mathcal{B}_{agg}$ as *constrained maximal expansion* of $\mathcal{B}_{ori}$. Such expansion specifies the priority of dimensions compared to the *maximal expansion*. The *constrained maximal expansion* for a potential box $\mathcal{B}_{agg}$ makes a maximal expansion along a predefined specific dimension (called *priority dimension*) in uni-direction first (Lines 6-11) and then $\mathcal{B}_{ori}$, as the center, expands to both the value decreasing and increasing directions as large as possible for the other dimensions (Lines 12-26). In this paper, we call the priority dimension d as *main sequence* and set the value decreasing direction as the one-way on d (Line 3). Let $i \in \{1, ..., n\}$ be a dimension. The *maximal expansion in one specified direction* along i of $\mathcal{B}_{ori}$ refers to traversing the align boxes from $\mathcal{B}_{ori}$ itself as start until the first box with *violated* prediction is encountered. The range of all traversals along *main sequence* is called the *maximum height* of the $\mathcal{B}_{agg}$ corresponding to the $\mathcal{B}_{ori}$ (Lines 7-10).



(a) Refined Boxes with State (b) B_{agg} for B_{ori1} (c) B_{agg} for B_{ori2} (d) Top View of (c)

(e) B_{agg} for $B_{ori3,5}$ (f) B_{agg} for B_{ori7} (g) B_{agg} for B_{ori4} (h) B_{agg} for B_{ori6}

Figure 4: Demonstration of Aggregation. $\mathcal{I}$ is divided into 2*3*3 sub-boxes for illustration. The blue boxes are potential while the pink ones are violated. The arrows denote the direction of the value increasing.

For example, Figure 4(a) shows $\mathcal{BS}_{ori} \in \mathbb{BS}^3$ made up with 7 predicted potential boxes $\mathcal{B}_{ori_1}$,...,$\mathcal{B}_{ori_7}$. We pick the blue axis as the main sequence d. As for $\mathcal{B}_{ori_1}$, it itself is already the end along d in the one-way direction, so the*maximum height* of $\mathcal{B}_{ori_1}$ is its own range on d. Because $\mathcal{B}_{ori_1}$ is surrounded by violation boxes within *maximum height*, the other dimensions could not make expansion any more, making $\mathcal{B}_{agg_1}$ be $\mathcal{B}_{ori_1}$ itself shown in Figure 4(b).

The construction of *constrained maximal expansion* will draw on the bottom-up dynamic programming to design the heuristic, so as to effectively reduce the computational overhead introduced by huge search space and repeated computation. After sorting all $\mathcal{B}_{ori}$ in the one-way direction on d (Lines 2-4), we could model the solution into dynamic programming from two perspectives. On the one hand, for the potential boxes only differing the range on main sequence d, when they respectively traverse to search for *maximum height*, the $\mathcal{B}_{ori}$ with larger value on d repeatedly visit the align boxes on the part of its *maximum height* overlapped with the $\mathcal{B}_{ori}$ having a smaller value on d. On the other hand, if the adjacent $\mathcal{B}_{A^2} \in \mathbb{B}^n$ with $[x_{\mathcal{B}_{A^2}}{}_l^d, x_{\mathcal{B}_{A^2}}{}_u^d]$ of $\mathcal{B}_{ori}$ with $[x_{\mathcal{B}_{ori}}{}_l^d, x_{\mathcal{B}_{ori}}{}_u^d]$ along d satisfying $x_{\mathcal{B}_{A^2}}{}_u^d = x_{\mathcal{B}_{ori}}{}_l^d$ is also potential (Line 6), then the *maximum height* of $\mathcal{B}_{ori}$ can be directly the sum of $\mathcal{B}_{A^2}$'s *maximum height* and itself projected on d without traversing again (Lines 7-8), and the expansions of all other dimensions except d of $\mathcal{B}_{ori}$ are limited by the expansion of $\mathcal{B}_{A^2}$. As Figure 4(d) explains, after determining the *maximum height* of $\mathcal{B}_{ori_2}$, although the longest traversing only considering red and green axes respectively are shown as the white arrows, the valid expansion along these two dimensions of $\mathcal{B}_{ori_2}$'s *constrained maximal expansion* is the black range, without beyond the scope of $\mathcal{B}_{A^2}$'s *constrained maximal expansion*. This implicitly keeps the *maximum height* found of $\mathcal{B}_{ori_2}$ still be the range of $\mathcal{B}_{ori_2}$'s *constrained maximal expansion* projected on d. Therefore, there is also a relationship between $\mathcal{B}_{A^2}$'s *constrained maximal expansion* and $\mathcal{B}_{ori}$ on all dimensions. The above two aspects separately reflect the overlapping sub-problems and optimal substructure properties of dynamic programming.

We further use a greedy strategy to optimize the operation with the expansion of other dimensions (Lines 12-26) after the operation on *main sequence*. We separately solve the expansion in both directions on each dimension while keeping *maximum height* of $\mathcal{B}_{ori}$ (Lines 12-22) and then span all dimensional intervals to be a *Box* hoping to directly make the new space as large as possible (Line 23), like Figure 4(e) to (h). We need to determine whether the new box is a potential candidate by checking whether the sub-boxes it covers are all potential as predicted before (Lines 24-26). Considering the restriction of $\mathcal{B}_{A^2}$, we make endpoints for each dimension align with the one closer to $\mathcal{B}_{ori}$ between the corresponding expansion of $\mathcal{B}_{A^2}$ and the longest traversing only considering single-dimension (Lines 12-22). Such greedy heuristic may introduce violation spaces not in $\mathcal{BS}_{ori}$; thus, we discard the invalid space like Figure 4(g) and (h). Our heuristic maintains the same potential space as $\mathcal{BS}_{ori}$, as in Proposition 3.2 through proof by contradiction and induction.

Proposition 3.2. *Our aggregation method exactly covers all the potential boxes as predicted before, that is,* $\mathcal{BS}_{agg} = \mathcal{BS}_{ori}$.

3.2.5 Ordered Verification. We temporarily keep the nested structure introduced by the optimal structure among $\mathcal{B}_{agg}$ of sorted $\mathcal{B}_{ori}$ on *main sequence* if existing, e.g. Figure 4(b) and (c), which is beneficial for making up for prediction errors.

We commence by enumerating the quantity of potential partitioned sub-boxes within each candidate $\mathcal{B}_{agg}$. Subsequently, we arrange them in descending order based on this count and systematically engage the verifier to furnish formal assurances. In the sequential process, if any larger one in the nested structure is verified to be the true precondition itself, then the other contained smaller candidates would be discarded without invoking the verifier anymore. If the larger potential one violates property, there are still smaller nested candidates implicitly representing part of a division of the larger failure candidate rather than deliberately starting the process for input splitting again. The candidates finally determined as property satisfaction by the verifier are the sufficient preconditions we pursue.

3.2.6 Overall. We summarize the whole inference process shown in Algorithm 2, and the details of each function work in concert with each section above in order.

Algorithm 2 Infer Non-Trivial Sufficient Precondition

```
Input: the desired property $\mathcal{P}$, DNN $f : \mathbb{R}^n \to \mathbb{R}^m$ and the input
    box domain $\mathcal{I}$.
Output: the sufficient Boxes precondition $\mathcal{OS}$
1: $\bar{f} \leftarrow$ buildingDNN($\mathcal{P}, f$)
2: $\mathbb{I} \leftarrow$ inputSplitting($domain, splittingMethod$)
3: $\mathbb{S} \leftarrow$ statePrediction($\mathbb{I}, \bar{f}$)
4: $\mathbb{C} \leftarrow$ aggregation($\mathbb{I}, \mathbb{S}$)
5: $\mathbb{Q} \leftarrow$ orderedVerify($\mathbb{C}, \mathcal{P}, f$)
    ▷ $\mathbb{Q} = \{(each\ c\ in\ \mathbb{C}, label)\}, label \in \{pass, unsafe, unknown\}$
6: $\mathcal{OS} \leftarrow \{(c, label) | (c, label) \in \mathbb{Q} \wedge label = pass\}$
```

3.3 Make Sufficient Precondition Weaker

The granularity of the input domain impacts state prediction and challenges the verifier's solving capability. As for the space partially containing preconditions, the finer the splitting is, the more precise the positioning of preconditions and violations will be. Coarse splitting may overlook the presence of preconditions inside, while excessively fine splitting will be computationally expensive. Thus, we propose to iterate on inferring preconditions as a compromise, progressively refining the division of specific space as needed.

We provide how to make sufficient precondition as weak as possible, that is, cover as much space as possible, based on the process shown in Algorithm 3. We take the maximum iteration times R as the termination condition for example, and can also adapt to others such as time or the lower limit of precondition coverage (Line 4). The featured DNN $\bar{f}$ only needs to be handled once because nothing relevant changes through iterations (Line 3). The only difference is the target input domains dealt with in each turn for precondition inference (Lines 5, 20). The iteration adaptively selects input splitting schemes as discussed in Section 3.2.2 for different domains (Lines 7-12). As for the initial input domain $\mathcal{I}$, we don't know whether it meets $\mathcal{P}$ or not (Line 2) and get no hint for it, so it's natural to make input splitting by *dimension splitting*. If the domain has any additional information, splitting with information is a tailored choice, e.g. counterexamples offered by the verifier. The subsequent processing of each domain is basically consistent with Section 3.2, except that we have conducted a detailed classification and application based on the results of the verifier (Lines 15-18). We sample N input points uniformly in each candidate without the *pass* label and then evaluate whether such a sub-domain still has the potential for further processing as an input domain for the next round. We count the number of samples meeting $\mathcal{P}$ among these N points denoted as PN and use their ratio $\omega = PN/N$ to represent the proportion of preconditions that may be ignored in such candidate discarded by the verifier. We finally choose the ones beyond our expected potential threshold δ to further participate in iteration later (Lines 17-18, 20).

4 Implementation and Evaluation

We have implemented our methods into a tool called *PreBoxes* and evaluated it with a series of experiments. Firstly, we discuss the experimental setup in Section 4.1. Next, we design a set of research questions to evaluate *PreBoxes* in Section 4.2 and present the corresponding results and discussions in Section 4.3. Finally, we discuss the limitation in Section 4.4.

4.1 Experimental Settings

4.1.1 Implement. We have implemented *PreBoxes* using Python under the framework of PyTorch to integrate with $\alpha\beta$-CROWN [1] as the verifier. We have conducted all the experiments on a machine with 12th Gen Intel Core i9-12900K, 128 GB DDR4 3200MHz RAM, and NVIDIA GeForce RTX 4090, running Windows 11 with subsystem for Linux.

[1]https://github.com/Verified-Intelligence/alpha-beta-CROWN

Algorithm 3 Synthesize Weaker Sufficient Precondition

```
Input: the desired property $\mathcal{P}$, DNN $f : \mathbb{R}^n \to \mathbb{R}^m$, the input
    box domain $\mathcal{I}$, maximum iterations $R$ and potential assessment
    threshold $\delta$.
Output: the sufficient Boxes precondition $\mathcal{OS}$
 1: $\mathcal{OS} \leftarrow \{\}$
 2: $\mathcal{IS} \leftarrow \{(\mathcal{I}, unknown)\}$
 3: $\bar{f} \leftarrow$ buildingDNN($\mathcal{P}, f$)
 4: while $iteration \le R$ do
 5:     $\mathcal{IS}_{temp} \leftarrow \{\}$
 6:     for each ($domain, label$) in $\mathcal{IS}$ do
 7:         if $label = unknown$ then
 8:             $splittingMethod \leftarrow dimensionSplitting$
 9:         else if $label = unsafe$ then
10:             $splittingMethod \leftarrow counterexampleSplitting$
11:         end if
12:         $\mathbb{I} \leftarrow$ inputSplitting($domain, splittingMethod$)
13:         $\mathbb{S} \leftarrow$ statePrediction($\mathbb{I}, \bar{f}$)
14:         $\mathbb{C} \leftarrow$ aggregation($\mathbb{I}, \mathbb{S}$)
15:         $\mathbb{Q} \leftarrow$ orderedVerify($\mathbb{C}, \mathcal{P}, f$) $\triangleright$ $\mathbb{Q} = \{(each\ c\ in\ \mathbb{C}, label)\}$,
            $label \in \{pass, unsafe, unknown\}$
16:         $\mathcal{OS} \leftarrow \mathcal{OS} \cup \{(c, label) | (c, label) \in \mathbb{Q} \land label = pass\}$
17:         $\mathcal{IS}_{pot} \leftarrow$ pick($\{(c, label) | (c, label) \in \mathbb{Q} \land (label =$
            $unknown \lor label = unsafe)\}, \delta$)
18:         $\mathcal{IS}_{temp} \leftarrow \mathcal{IS}_{temp} \cup \mathcal{IS}_{pot}$
19:     end for
20:     $\mathcal{IS} \leftarrow \mathcal{IS}_{temp}$
21: end while
```

4.1.2 Competitors. We consider three typical under-approximation schemes for precondition synthesis, namely *Uniform* [19], *NonUniform* [19] and *PreimageAppro* [36]. *Uniform* and *NonUniform* implement the ideas of solving certified adversary-free regions as large as possible separately in the form of uniform and non-uniform bounds across all input features respectively [19]. Both of them rely on incomplete verification algorithm [35] and provide sufficient preconditions as the union of all adversary-free regions around massive inputs in the format of *Boxes* as well. Following both the incomplete [32] and complete [30] verification algorithms, *PreimageAppro* [36] takes sufficient precondition as the goal in a less restricted way to adapt these verification works and optimize on the whole bounded input domain. *PreimageAppro* provides the results in the format of a disjoint union of polytopes. However, the methods above solely synthesize preconditions for desired properties representing a convex output set as conjunctions of linear constraints, referred to as *simple properties* in this paper.

4.1.3 Benchmark. We gather four benchmarks as shown in Table 1. We use the same index of *ACASXu* models as VNN-COMP [2] abbreviating as *nn_i_j*, where $i \in \{1, 2, 3, 4, 5\}$, $j \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and use *NN4Sys* models *mscn_128d* and *mscn_128d_dual*. As far as we know, we accomplish precondition synthesis on a *Non-ReLU complex structured network* like *NN4Sys*, a multi-set feed-forward network [14] where tables, joins, and predicates are represented as separate modules with concatenation operations. The number of effective input dimensions is marked as $\sharp Dim$.

We have set rich trustworthiness properties for different benchmarks, as shown in Table 6. Beyond *simple properties* considered in the previous work, we introduce all *complex properties* in [13] (original indices provided in parentheses). Although *ACASXu* has been discussed in the previous work, *PreBoxes* does not fix the numerical value of any input features and completes the precondition synthesis with respect to *complex properties*.

4.1.4 Evaluation Metrics. Firstly, we use the metric the geometric average diameter $\epsilon = \sum_{i=1}^{d}(x_i^u - x_i^l)/d$ in [19] to evaluate the weakness extent of the precondition when concerning robustness. ϵ_{avg} represents the mean diameter of all *Box* preconditions in *Boxes*, and ϵ_{max} is the maximum diameter among *Boxes*.

Secondly, we use the metric in [36] as a normalized measure for assessing the quality of the sufficient precondition, that is the coverage ratio R_{cov} to be the ratio of volume covered by *precondition solutions* to the volume of the *exact precondition*. The estimation of the *exact precondition*'s volume $vol(ep)$ is consistent with [36] using the input domain $\mathcal{I}$ relevant to property P as $vol(ep) \approx vol(\mathcal{I}) \times (\sum_{i=1}^{N} 1_{f(x_i) \in \mathcal{P}}/N)$, where $x_1, ...x_N$ are samples from $\mathcal{I}$. Further, $R_{cov} \approx (\sum_{i=1}^{N} 1_{f(x_i) \in \mathcal{OS}}/N)/(\sum_{i=1}^{N} 1_{f(x_i) \in \mathcal{P}}/N)$ is defined in practice, where $\mathcal{OS}$ presents the synthesis solution.

4.2 Research Questions

To evaluate the effectiveness, utility, and factors of *PreBoxes*, we consider the following research questions.

RQ1(**Effectiveness**): *Can PreBoxes solve the non-trivial sufficient preconditions? How effective is PreBoxes in synthesizing weaker sufficient preconditions compared to the prior approaches? Beyond that, what functionality does PreBoxes contribute and how does it perform?*

Due to the support of the verifier in the *check* phase, the correctness of any synthetic preconditions of *PreBoxes* is naturally guaranteed if existing. We have designed various scenes to assess the effectiveness of *PreBoxes* as a whole from two perspectives: scenes that can be completed by existing works and functions that *PreBoxes* expands. At first, we make comparisons with three typical methodologies mentioned in Section 4.1.2 to show the weaknesses degree of sufficient precondition results as well as the ability to cope with different sizes of input domains within an effective time. Then, we separately demonstrate its competitive capabilities to handle *complex properties* and *Non-ReLU complex structured networks* in Section 4.1.3.

RQ2(**Utility**): *Does each functional module in PreBoxes contribute to continuously reducing the overhead of verifier calls? Is the assumption used in the design of state prediction reasonable? Does iterative running of PreBoxes help obtain weaker sufficient preconditions?*

We evaluate the utility of functional modules in *PreBoxes* on Algorithm 2 and Algorithm 3 from three perspectives. Firstly, the results of modules *Input Splitting*, *State Prediction*, and *Aggregation* in the *guess* phase can theoretically be passed directly to the *check* phase for verification. They skip subsequent steps as they provide different sets of precondition candidates. However, their interrelationship is designed to reduce the cost caused by large numbers of verifications. We evaluate whether such expectation holds. Secondly, the subsequent modules of *State Prediction* only consider

[2]https://github.com/ChristopherBrix/vnncomp2022_benchmarks

Table 1: DNN Benchmark for Precondition Synthesis

Benchmark	♯ Model	Source	Application	Network Types	♯ Neurons	♯ Dim
Syn	1	[19]	Classification	FC.+ReLU	20	2
CartPole	1	[22]	Reinforcement Learning	FC.+ReLU	128	4
ACASXu	17	[22]	Airborne Collision Avoidance	FC.+ReLU	300	5
NN4Sys	2	[22]	Cardinality Estimation Part	Complex(ReLU+Sigmoid)	1024-2048	4-6

Table 2: Desired Properties for Precondition Synthesis

Benchmark	Property Index	Property Meaning	Source	Property Types	Simple
Syn	$prop_i, i \in \{1, ..., 10\}$	the network will predict label i	[19]	Robustness	✓
CartPole	$prop_1$	push cart to the left	[22]	Safety	✓
CartPole	$prop_2$	push cart to the right	[22]	Safety	✓
ACASXu	$prop_1$	the score for weak left will be maximal	Custom	Safety	✓
ACASXu	$prop_2$	the score for weak right will be maximal	Custom	Safety	✓
ACASXu	$prop_3$	the score for strong left will be maximal	Custom	Safety	✓
ACASXu	$prop_4$	the score for strong right will be maximal	Custom	Safety	✓
ACASXu	$prop_5$(3/4)	the score for COC will not be minimal	[13]	Safety	✗
ACASXu	$prop_6$(7)	the scores for strong right and strong left will be never the minimal scores	[13]	Safety	✗
ACASXu	$prop_7$(8)	the network will either output COC or continue advising weakleft	[13]	Safety	✗
ACASXu	$prop_8$(2)	the score for COC will not be maximal	[13]	Safety	✗
ACASXu	$prop_9$(6/10)	the score for COC will be minimal	[13]	Safety	✓
NN4Sys	$prop_1$	the score will always be in a certain range	Custom	Safety	✓
NN4Sys	$prop_2$	the score will always be below a certain threshold	[22]	Safety	✓

sub-boxes predicted to be potential. There is an assumption underlying that it is better to consider boxes predicted as potential rather than unsafe. Here, we tend to define *better* as *how close the sub-box is to the true precondition*, meaning the actual coverage of the true precondition in potential boxes is greater than that in unsafe boxes. The first two perspectives are related to both Algorithm 2 and Algorithm 3. Finally, we also verify whether Algorithm 3 helps weaken results based on Algorithm 2.

RQ3(Factors): *How do different hyper-parameter settings affect the effectiveness of PreBoxes?*

According to the experimental settings on iteration in RQ2, it is obvious that the fineness of *Input Splitting* and the potential assessment threshold δ in *iterative running* affect the weakness of results. To avoid redundant description, we evaluate the impact of the solving capability of *verifier* and the density of boundary sampling in *State Prediction.*

4.3 Results and Analysis

4.3.1 Answers to RQ1. We first separately compare *PreBoxes* with the competitors in Section 4.1.2 and then display its competitive capabilities on complex properties and DNNs in the Section 4.1.3.

Comparison with Uniform and NonUniform: We compare *PreBoxes* with *Uniform* and *NonUniform* on *Syn* and *ACASXu* to show their non-trivial sufficient preconditions on *simple properties* because both competitors can only apply such properties. We maintain the default parameter configuration of competitors without a time limit. As for *PreBoxes*, we set a 100s timeout with 10^4 input-splitting sub-boxes on *Syn* and a 10 min timeout with 5^5 input-splitting sub-boxes on *ACASXu.*

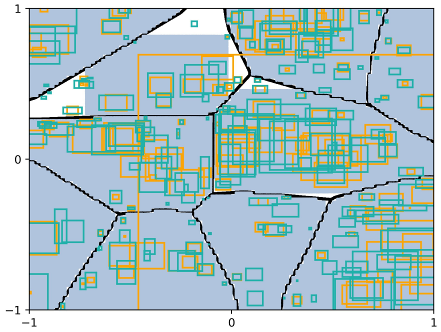


Figure 5: Comparison with Uniform and NonUniform on Syn

The results for $prop_i$, $i \in \{1, ..., 10\}$ on *Syn* are shown in Figure 5. The orange rectangles represent the *Boxes* covered by the sufficient precondition found by *Uniform* and green corresponds to *NonUniform.* The blue region depicts the Boxes solution of *PreBoxes.* The black lines depict *decision boundary* solved by brute force search. Firstly, *PreBoxes* covers a significantly larger area in the entire input domain compared to both *Uniform* and *NonUniform*, indicating its efficient synthesis of weaker sufficient preconditions than others. Secondly, although some preconditions solved by *Uniform* or *NonUniform* are not included in the solutions of *PreBoxes*, we observe that *PreBoxes* offers a better correctness guarantee. We observed an orange rectangle spanning multiple black boundaries, indicating that a *Box* precondition solved by *Uniform* for a specific robustness property incorrectly covers multiple preconditions related to other properties. However, thanks to the *check* phase, *PreBoxes* will never synthesize incorrect preconditions.

Table 3 shows the results of $prop_1$-$prop_4$ on *ACASXu*. The average time for each solution is 761.3s for *Uniform* and 710.7s for *NonUniform*, while we set 600s timeout for *PreBoxes*. On the one hand, *PreBoxes* solves much weaker sufficient preconditions in general than both *Uniform* and *NonUniform*. Based on ϵ_{avg} and ϵ_{max} in Section 4.1.4, $R_{max} = \epsilon_{max}^{PreBoxes}/max\left(\epsilon_{max}^{Uniform}, \epsilon_{max}^{NonUniform}\right)$ and $R_{avg} = \epsilon_{avg}^{PreBoxes}/max\left(\epsilon_{avg}^{Uniform}, \epsilon_{avg}^{NonUniform}\right)$ are defined here for clearness. Whether it is the overall situation of *Boxes* as R_{avg} or individual situations as R_{max}, they are all greater than 1 when being able to compare. On the other hand, *PreBoxes* shows better stability on *ACASXu*. Both *Uniform* and *NonUniform* have NaN solutions, which means that there is a numerical error in results, i.e., the method failed for a certain instance.

Comparison with PreimageAppro: We compare *PreBoxes* with the *PreimageAppro* on *ACASXu* and *CartPole* to show their precondition synthesis abilities also on *simple properties* due to the limitation of *PreimageAppro*. Both *PreBoxes* and *PreimageAppro* synthesize preconditions within a given initial input domain.

As for *ACASXu*, we use the same instances listed in Table 3 and set coverage ratio as the metric. The experimental setup of *PreBoxes* is the same as before. *PreimageAppro* is set to the same time threshold as *PreBoxes*, and both have the same initial input domain. As shown in Table 4, although *PreimageAppro* solves weaker preconditions than *PreBoxes* in some instances, in most cases *PreimageAppro* cannot solve any answer at all. *PreBoxes* has a relatively stable solving ability.

Table 4: Comparison R_{cov} with PreimageAppro on ACASXu

Model	Property	*PreBoxes*	*PreimageAppro*
nn_1_1	$prop_1$	0.2828	0.5006
nn_1_7	$prop_1$	0.1954	0.0
nn_1_5	$prop_2$	0.3047	0.6265
nn_1_8	$prop_2$	0.3403	0.5018
nn_1_2	$prop_3$	0.0613	0.0
nn_1_4	$prop_3$	0.0467	0.0
nn_1_3	$prop_4$	0.3337	0.0
nn_2_8	$prop_4$	0.2678	0.0

As for *CartPole*, we first initialize two different ranges of input domains: the *Small* is consistent with the default setting of *PreimageAppro*[3], which is $((-0.5, 0.5), (-1, 1), (-0.1, 0.1), (-1, 1))$, and the *Large* is instantiated from [9], which is $((-4.8, 4.8), (-100, 100), (-0.418, 0.418), (-100, 100))$. *Large* is *401.28K* times *Small*. We keep default configurations of *PreimageAppro*. As for *PreBoxes*, we set input-splitting granularity to 6^4 for both domain types for convenience. When *PreimageAppro* offers solutions, we align the running time of *PreBoxes* with *PreimageAppro* and compare R_{cov}. If *PreimageAppro* fails to provide results by the default timeout, no runtime is set for *PreBoxes*.

We found that *PreimageAppro*'s ability to synthesize preconditions is related to the size of the input domain, while *PreBoxes* generally shows good stability. As shown in Table 5, *PreimageAppro* cannot offer any results until timeout under the *Large* , but it shows a relatively faster convergence than *PreBoxes* on the *Small*. This is because the size of the domain will directly affect the tightness of *PreimageAppro*'s approximate optimization of preconditions through linear relaxation. Even if using global branching, large domains still make the resulting bounds relatively loose. In *PreBoxes*, the domain partitioned by input splitting or iteration transfers to effective candidates through state prediction and aggregation.

Table 5: Comparison with PreimageAppro on CartPole

Property	Domain	Competitors	R_{cov}	Time(s)
$prop_1$	Large	*PreimageAppro*	0.0	2130.866
		PreBoxes	0.805	20.513
$prop_1$	Small	*PreimageAppro*	0.752	80.747
		PreBoxes	0.602	81.374
$prop_2$	Large	*PreimageAppro*	0.0	2691.926
		PreBoxes	0.782	16.392
$prop_2$	Small	*PreimageAppro*	0.754	19.625
		PreBoxes	0.335	20.314

Dealing with Complex Properties on Huge Domain: As shown in Table 6, we evaluate *PreBoxes* with all such *complex properties* in [13] on the huge domain of *ACASXu* sourced from [22]. The domain here $((0, 60760), (-\pi, \pi), (-\pi, \pi), (0, 1200), (0, 1200))$ is about $10.76M$ times *Large*. We set 3600s as a timeout for each instance.

Table 6: Sufficient Precondition Synthesis for Complex Properties on Huge Domain of ACASXu

Property	Model	R_{cov}	ϵ_{avg}	ϵ_{max}
$prop_5$	nn_1_6	0.013	1527.166	2205.543
$prop_5$	nn_5_2	0.008	1619.726	2185.542
$prop_6$	nn_4_3	0.402	8980.456	11113.629
$prop_6$	nn_3_1	0.076	8998.609	9504.942
$prop_7$	nn_2_3	0.376	9507.988	11113.942
$prop_7$	nn_4_2	0.056	9028.138	11083.472
$prop_8$	nn_2_6	0.469	9338.675	12602.472
$prop_8$	nn_3_1	0.430	9672.194	11113.942

As Table 6 shows, *PreBoxes* offers provable sufficient preconditions for each instance and has the capability of directly dealing

[3]https://github.com/Zhang-Xiyue/PreimageApproxForNNs

Table 3: Comparison with Uniform and NonUniform on ACASXu

Model	Property	Uniform		NonUniform		PreBoxes		R_{avg}	R_{max}
		ϵ_{avg}	ϵ_{max}	ϵ_{avg}	ϵ_{max}	ϵ_{avg}	ϵ_{max}		
*nn*_1_1	$prop_1$	NaN	4230.669	4209.0618	4277.2188	7090.2351	10154.1026	1.6845	2.4001
*nn*_1_7	$prop_1$	4477.736	5087.5273	4637.2815	5368.5503	8095.5597	10106.1026	1.7458	1.8825
*nn*_1_5	$prop_2$	NaN	4824.229	4382.5498	5078.711	7038.2416	10154.1026	1.6060	1.9993
*nn*_1_8	$prop_2$	NaN	4208.251	4208.2491	4208.284	8423.7711	10154.1027	2.0017	2.4129
*nn*_1_2	$prop_3$	NaN	4237.2437	NaN	4251.794	10058.6053	10058.6053	NaN	2.3657
*nn*_1_4	$prop_3$	NaN	4248.6064	4213.2214	4325.551	7628.2053	7628.2053	1.8105	1.7635
*nn*_1_3	$prop_4$	NaN	5034.0156	4671.1998	5388.9443	7540.2427	10202.1027	1.6142	1.8932
*nn*_2_8	$prop_4$	NaN	NaN	NaN	5794.353	10154.6053	10154.6053	NaN	1.7525

Table 7: Sufficient Precondition Synthesis on NN4Sys

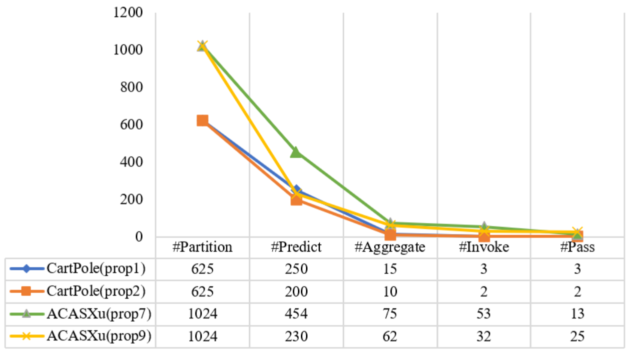
Model	Property	♯ Dim	R_{cov}	Time(s)
mscn_128d	$prop_1$	4	0.999	8.598
mscn_128d	$prop_1$	5	0.989	35.777
mscn_128d	$prop_1$	6	0.931	182.961
mscn_128d_dual	$prop_2$	4	0.701	150.682
mscn_128d_dual	$prop_2$	5	0.606	503.327
mscn_128d_dual	$prop_2$	6	0.552	1775.867

with such properties as formulas over linear inequalities supporting mixed conjunctive and disjunctive form. Although R_{cov} of our method within the time threshold is relatively low for some instances due to the overhead balance caused by the huge domain, we observe that the solved range of provable preconditions is still considerably non-trivial based on ϵ_{avg} and ϵ_{max}.

Dealing with Non-ReLU Complex Structured DNNs: We select *NN4Sys* to examine the capability of *PreBoxes* for dealing with Non-ReLU DNN. We formulate the input domain as well as property type with reference to the settings of VNN-COMP and randomly generate variable input dimensions based on ♯*Dim*. We set the initial splitting granularity of $5^{\sharp Dim}$ for each instance in Table 7. To the best of our knowledge, no previous work has achieved such capability. As shown in Table 7, our method can still efficiently solve provable sufficient preconditions with considerable coverage, even for networks with complex structures, thousands of parameters, and non-ReLU activation functions.

4.3.2 Answers to RQ2. We evaluate the utility of functional modules in *PreBoxes* as follows.

Functional Certificate: If *PreBoxes* terminates at a specific step, the number of boxes generated here is the number of verifier calls with skipping the next steps. We conducted experiments on two benchmarks: *CartPole* (evaluated with $prop_1$ and $prop_2$) and *ACASXu* (evaluated with *nn*_2_3 using $prop_7$ and *nn*_2_4 using $prop_9$). As Figure 6 shows, the interrelationship design of each functional module contributes to continuously reducing the overhead of verifier calls where the first four columns represent the number of candidate boxes in *Input Splitting*, *State Prediction*, *Aggregation* and *Ordered Verification* respectively with ablation, and the last column represents the number of provable boxes passed by the verifier.



	#Partition	#Predict	#Aggregate	#Invoke	#Pass
CartPole(prop1)	625	250	15	3	3
CartPole(prop2)	625	200	10	2	2
ACASXu(prop7)	1024	454	75	53	13
ACASXu(prop9)	1024	230	62	32	25

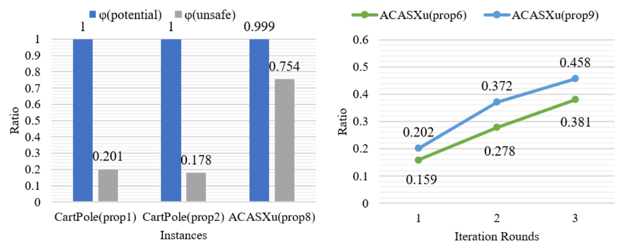
Figure 6: Ablation Evaluation of Functional Certificate

Assumption Justification: In practice, we define $\varphi(potential)$ and $\varphi(unsafe)$ separately as the mean of all R_{cov} of sub-boxes predicted as the according state, which evaluates the average degree of closeness to preconditions themselves. We conducted experiments on two benchmarks: *CartPole* (evaluated with $prop_1$ and $prop_2$) and *ACASXu* (evaluated with *nn*_5_2 using $prop_8$). As shown in Figure 7(a), $\varphi(potential)$ is generally greater than $\varphi(unsafe)$. Therefore, even if the sub-boxes predicted as unsafe are discarded, *PreBoxes* can obtain a reasonable cost-effective benefit.

Iterative Running: Take *ACASXu* evaluated with *nn*_3_1 separately using $prop_6$ and $prop_9$ as examples. We set each instance to iterate three times and obtain R_{cov} of the provable sufficient preconditions for each round. As shown in Figure 7(b), the upward trend of lines shows that iterative running helps to cover more candidates, indicating weaker preconditions.

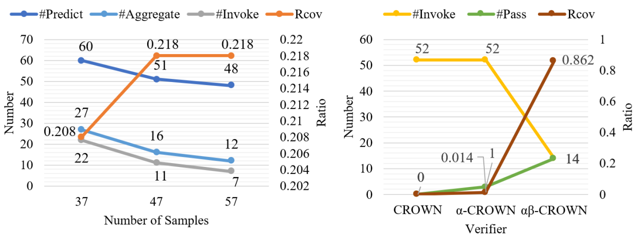
4.3.3 Answers to RQ3. Hyper-parameter evaluations are as follows.

Density of Boundary Sampling: We take *ACASXu* instance with *nn*_2_2 and $prop_8$ as examples and other instances show similar patterns. Except for the vertices of the sub-box, we separately uniformly sample 1, 3, and 5 points on each edge to assist in prediction (The total is 37, 47, and 57 respectively). With the same input splitting, we compare the number of candidates generated by each subsequent module as shown in Figure 8(a). Through the downward trend of ♯*Predict*, ♯*Aggregate*, and ♯*Invoke*, the increase in



(a) Assumption Support (b) Iterative Running

Figure 7: Assumption Justification and Iterative Running



(a) Density of Boundary Sampling (b) Solving Capability of Verifier

Figure 8: Impact of Typical Hyper-Parameter Settings

the number helps to improve the accuracy of prediction, thereby reducing redundant operation overhead for subsequent functional modules due to prediction mistakes. From the rise to the gentle trend of R_{cov}, appropriately increasing the density will help find weaker results, but there will be an inflection point helping the trade-off.

Solving Capability of Verifier: We use three representative verification algorithms as the verifier, and other settings are the same. They are the incomplete verification algorithms CROWN [35], α-CROWN [32], and the complete verification algorithm $\alpha\beta$-CROWN [30]. We take *CartPole* instance $prop_2$ as an example shown in Figure 8(b). The verifier with strong capabilities effectively reduces overhead and helps prove weaker preconditions under the same operation.

4.4 Limitations

The first limitation is that we work on the bounded input domain. Second, although we have made efforts such as aggregation to reduce the cost caused by the dimension disaster, scalability is still needed to justify and improve.

5 Related Work

Precondition synthesis has already been studied in program analysis and verification [2, 4, 15, 18, 33]. Recently, precondition synthesis for DNN is gradually attracting attention for developing trustworthy learning-enabled systems. Pasareanu et al. evaluate the DNN module's expectation within the safety framework, focusing on traditional program modules rather than directly analyzing the network itself [25]. Ahmed et al. adhere to Dijkstra's predicate transformer semantics rules to derive preconditions with abstracting DNNs [1]. Naik and Nuzzo introduce robustness contracts for compositional specification in cyber-physical systems with DNN components [23].

Inspired by the maximum local robust radius solution in DNN verification work [16, 28, 35], Liu et al. adapt the original verification algorithm to transform the optimizing objectives from uniform to non-uniform space [19]. Gopinath et al. analyze feed-forward DNNs layer by layer, proposing to extract patterns from neuron decisions as preconditions [10]. However, it is difficult to expand large-scale networks in this way. Some researchers further combine and optimize both incomplete and complete verification methods for precondition generation. Zhang et al. use input and ReLU splitting to generate sufficient under-approximation preconditions in the format of disjoint union of polytopes representation [36]. Kotha et al. make a convex over-approximation over the necessary precondition of DNNs in the format of a linearly constrained output set [17]. There are also researches focusing on exact preimage generation [21, 24]. However, such an exact solution is intractable, taking time exponentially in the number of neurons in neural networks. So far, most of these works often support solving piecewise linear DNNs and are restricted by the type of activation functions and properties as a convex output set in the form of conjunctions of linear constraints.

Moreover, there are a few existing works on splitting input regions to quantify robustness and fairness properties. Biswas et al. partition attributes based on whether each attribute is relaxed or the threshold size [5]. Bunel et al. design to split the input domain in half along its largest input feature range [6]. Wang et al. regard the gradient information as the influence on the output and pick the largest axis as the first to bisect [29]. Most of them obtain sub-regions waiting for independent verification at once. Our guess phase provides a process of first going from complete to fragmented, but then aggregating in a targeted manner to achieve partial completeness.

6 Conclusion

We propose a guess-and-check framework *PreBoxes* to synthesize sufficient Boxes preconditions for DNN concerning the given postcondition. *PreBoxes* under-approximates the exact precondition in a gray-box way, i.e. the *guess* phase is black-box-based for efficient generation of candidates, while the *check* phase is white-box-based formal verification. *PreBoxes* shows its competitive capabilities compared with three representative existing schemes when evaluated across four benchmarks. Moreover, *PreBoxes* expands functionality for complex properties and DNNs. In future research, we would like to explore more expressive forms of precondition to provide technical support for the security and interpretability of DNNs.

Data Availability

The source code of *PreBoxes* with the experimental data is available at Zenodo [20].

Acknowledgments

We thank the ISSTA 2024 reviewers for their constructive feedback. This work is supported by the National Key R&D Program of China (No.2022YFA1005101) and the National Natural Science Foundation of China (No.62032024).

References

[1] Shibbir Ahmed, Hongyang Gao, and Hridesh Rajan. 2024. Inferring Data Preconditions from Deep Learning Models for Trustworthy Prediction in Deployment. In *ICSE'2024: The 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Lisbon, Portugal).

[2] Angello Astorga, P. Madhusudan, Shambwaditya Saha, Shiyu Wang, and Tao Xie. 2019. Learning stateful preconditions modulo a test generator. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA) *(PLDI 2019)*. Association for Computing Machinery, New York, NY, USA, 775–787. https://doi.org/10.1145/3314221.3314641

[3] Mark R. Baker and Rajendra B. Patil. 1998. Universal Approximation Theorem for Interval Neural Networks. *Reliable Computing* 4, 3 (01 Aug 1998), 235–239. https://doi.org/10.1023/A:1009951412412

[4] Julien Bertrane, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, and Xavier Rival. 2015. Static Analysis and Verification of Aerospace Software by Abstract Interpretation. *Found. Trends Program. Lang.* 2, 2–3 (dec 2015), 71–190. https://doi.org/10.1561/2500000002

[5] Sumon Biswas and Hridesh Rajan. 2023. Fairify: Fairness Verification of Neural Networks. In *Proceedings of the 45th International Conference on Software Engineering* (Melbourne, Victoria, Australia) *(ICSE '23)*. IEEE Press, 1546–1558. https://doi.org/10.1109/ICSE48619.2023.00134

[6] Rudy Bunel, Ilker Turkaslan, Philip H.S. Torr, Pushmeet Kohli, and M. Pawan Kumar. 2018. A Unified View of Piecewise Linear Neural Network Verification. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 4795–4804.

[7] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. 2018. End-to-End Driving Via Conditional Imitation Learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)* (Brisbane, Australia). IEEE Press, 1–9. https://doi.org/10.1109/ICRA.2018.8460487

[8] Marcelo Forets and Christian Schilling. 2023. *The Inverse Problem for Neural Networks*. Springer Nature Switzerland, 241–255. https://doi.org/10.1007/978-3-031-46002-9_14

[9] Farama Foundation. 2022. *Gym Documentation*. https://www.gymlibrary.dev/environments/classic_control/cart_pole/

[10] Divya Gopinath, Hayes Converse, Corina S. Păsăreanu, and Ankur Taly. 2020. Property inference for deep neural networks. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering* (San Diego, California) *(ASE '19)*. IEEE Press, 797–809. https://doi.org/10.1109/ASE.2019.00079

[11] Arie Gurfinkel and Sagar Chaki. 2010. Boxes: A Symbolic Abstract Domain of Boxes. In *Static Analysis*, Radhia Cousot and Matthieu Martel (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 287–303.

[12] Anan Kabaha and Dana Drachsler-Cohen. 2023. Maximal Robust Neural Network Specifications via Oracle-Guided Numerical Optimization. In *Verification, Model Checking, and Abstract Interpretation*, Cezara Dragoi, Michael Emmi, and Jingbo Wang (Eds.). Springer Nature Switzerland, Cham, 203–227.

[13] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Computer Aided Verification*, Rupak Majumdar and Viktor Kunčak (Eds.). Springer International Publishing, Cham, 97–117.

[14] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. 2018. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. arXiv:1809.00677 [cs.DB]

[15] Marko Kleine Büning, Johannes Meuer, and Carsten Sinz. 2022. Refined Modularization for Bounded Model Checking Through Precondition Generation. In *Formal Methods and Software Engineering: 23rd International Conference on Formal Engineering Methods, ICFEM 2022, Madrid, Spain, October 24–27, 2022, Proceedings* (Madrid, Spain). Springer-Verlag, Berlin, Heidelberg, 209–226. https://doi.org/10.1007/978-3-031-17244-1_13

[16] J. Zico Kolter and Eric Wong. 2017. Provable defenses against adversarial examples via the convex outer adversarial polytope. *CoRR* abs/1711.00851 (2017). arXiv:1711.00851 http://arxiv.org/abs/1711.00851

[17] Suhas Kotha, Christopher Brix, Zico Kolter, Krishnamurthy Dvijotham, and Huan Zhang. 2023. Provably Bounding Neural Network Preimages. arXiv:2302.01404 [cs.LG]

[18] Jonas Krämer, Lionel Blatter, Eva Darulova, and Mattias Ulbrich. 2022. Inferring Interval-Valued Floating-Point Preconditions. In *Tools and Algorithms for the Construction and Analysis of Systems: 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2–7, 2022, Proceedings, Part I* (Munich, Germany). Springer-Verlag, Berlin, Heidelberg, 303–321. https://doi.org/10.1007/978-3-030-99524-9_16

[19] Chen Liu, Ryota Tomioka, and Volkan Cevher. 2019. On Certifying Non-uniform Bound against Adversarial Attacks. *CoRR* abs/1903.06603 (2019). arXiv:1903.06603 http://arxiv.org/abs/1903.06603

[20] Zengyu Liu. 2024. Synthesizing Boxes Preconditions for Deep Neural Networks. https://doi.org/10.5281/zenodo.12673450

[21] Kyle Matoba. 2020. Exact Preimages of Neural Network Aircraft Collision Avoidance Systems. https://api.semanticscholar.org/CorpusID:231854560

[22] Mark Niklas Müller, Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T. Johnson. 2023. The Third International Verification of Neural Networks Competition (VNN-COMP 2022): Summary and Results. arXiv:2212.10376 [cs.LG]

[23] Nikhil Naik and Pierluigi Nuzzo. 2020. Robustness Contracts for Scalable Verification of Neural Network-Enabled Cyber-Physical Systems. In *2020 18th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*. 1–12. https://doi.org/10.1109/MEMOCODE51338.2020.9315118

[24] Théo Nancy, Vassili Maillet, and Johann Barbier. 2022. An Analytical Approach to Compute the Exact Preimage of Feed-Forward Neural Networks. arXiv:2203.00438 [cs.LG]

[25] Corina Pasareanu, Ravi Mangal, Divya Gopinath, and Huafeng Yu. 2023. Assumption Generation for the Verification of Learning-Enabled Autonomous Systems. arXiv:2305.18372 [cs.AI]

[26] Corina S. Păsăreanu, Ravi Mangal, Divya Gopinath, and Huafeng Yu. 2023. Assumption Generation for Learning-Enabled Autonomous Systems. In *Runtime Verification: 23rd International Conference, RV 2023, Thessaloniki, Greece, October 3–6, 2023, Proceedings* (Thessaloniki, Greece). Springer-Verlag, Berlin, Heidelberg, 3–22. https://doi.org/10.1007/978-3-031-44267-4_1

[27] D. Seto, B. Krogh, L. Sha, and A. Chutinan. 1998. The Simplex architecture for safe online control system upgrades. In *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No.98CH36207)*, Vol. 6. 3504–3508 vol.6. https://doi.org/10.1109/ACC.1998.703255

[28] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. 2018. Fast and Effective Robustness Certification. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2018/file/f2f446980d8e971ef3da97af089481c3-Paper.pdf

[29] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal security analysis of neural networks using symbolic intervals. In *Proceedings of the 27th USENIX Conference on Security Symposium* (Baltimore, MD, USA) *(SEC'18)*. USENIX Association, USA, 1599–1614.

[30] 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 complete and incomplete neural network verification. *Advances in Neural Information Processing Systems* 34 (2021).

[31] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. 2020. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems* 33 (2020).

[32] 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 Representations*. https://openreview.net/forum?id=nVZtXBI6LNn

[33] Xuejun Yang, Ji Wang, and Xiaodong Yi. 2010. Slicing Execution with Partial Weakest Precondition for Model Abstraction of C Programs. *Comput. J.* 53, 1 (2010), 37–49. https://doi.org/10.1093/comjnl/bxn075

[34] Sangdoo Yun, Jongwon Choi, Youngjoon Yoo, Kimin Yun, and Jin Young Choi. 2017. Action-Decision Networks for Visual Tracking with Deep Reinforcement Learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1349–1358. https://doi.org/10.1109/CVPR.2017.148

[35] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. 2018. Efficient Neural Network Robustness Certification with General Activation Functions. *Advances in Neural Information Processing Systems* 31 (2018), 4939–4948. https://arxiv.org/pdf/1811.00866.pdf

[36] Xiyue Zhang, Benjie Wang, and Marta Kwiatkowska. 2024. Provable Preimage Under-Approximation for Neural Networks. In *Tools and Algorithms for the Construction and Analysis of Systems: 30th International Conference, TACAS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6–11, 2024, Proceedings, Part III* (Luxembourg City, Luxembourg). Springer-Verlag, Berlin, Heidelberg, 3–23. https://doi.org/10.1007/978-3-031-57256-2_1

Received 2024-04-12; accepted 2024-07-03