Verification of Recurrent Neural Networks with Star Reachability

Hoang-Dung Tran
dtran30@unl.edu
University of Nebraska-Lincoln
Lincoln, Nebraska, USA

Sungwoo Choi
schoi9@huskers.unl.edu
University of Nebraska-Lincoln
Lincoln, Nebraska, USA

Xiaodong Yang
xiaodong.yang@vanderbilt.edu
Vanderbilt University
Nashville, Tennessee, USA

Tomoya Yamaguchi
tomoya.yamaguchi@woven-planet.global
Woven Planet
Tokyo, Japan

Bardh Hoxha
bardh.hoxha@toyota.com
Toyota NA R&D
Ann Arbor, Michigan, USA

Danil Prokhorov
danil.prokhorov@toyota.com
Toyota NA R&D
Ann Arbor, Michigan, USA

ABSTRACT

The paper extends the recent star reachability method to verify the robustness of recurrent neural networks (RNNs) for use in safety-critical applications. RNNs are a popular machine learning method for various applications, but they are vulnerable to adversarial attacks, where slightly perturbing the input sequence can lead to an unexpected result. Recent notable techniques for verifying RNNs include unrolling, and invariant inference approaches. The first method has scaling issues since unrolling an RNN creates a large feedforward neural network. The second method, using invariant sets, has better scalability but can produce unknown results due to the accumulation of overapproximation errors over time. This paper introduces a complementary verification method for RNNs that is both sound and complete. A relaxation parameter can be used to convert the method into a fast overapproximation method that still provides soundness guarantees. The method is designed to be used with NNV, a tool for verifying deep neural networks and learning-enabled cyber-physical systems. Compared to state-of-the-art methods, the extended exact reachability method is 10× faster, and the overapproximation method is 100× to 5000× faster.

KEYWORDS

Recurrent Neural Networks, Reachability Analysis, Verification

1 INTRODUCTION

A recurrent neural network (RNN) [20] is a particular neural network that supports modeling of sequential or time-series data. RNNs are designed to store information for periods of time and can learn complex patterns in data. The stored information influences the current input and output. Consequently, RNNs are well-suited for ordinal or temporal problems such as natural language processing [9, 18], speech recognition [4], and sensor/engine health prediction [5, 21]. Although RNNs are a powerful tool to address many complex, real-world challenges, they are also vulnerable to adversarial attacks. A slight and careful change in the input can fool an accurate RNN into an unexpected classification error [16]. To deploy RNNs in safety-critical applications, we need methods that formally verify their correctness.

Although formal verification of deep neural networks has received significant attention recently, most current works focus on FFNNs and CNNs [11, 31, 33]. Only a few papers deal with RNNs [1, 12, 23, 35] despite their extensive usage in practice. Verifying an RNN is challenging due to the “memory units” of the network. Unrolling is the first well-known approach [1]. This approach is not scalable as the size of the unrolled network quickly becomes too large to verify as the number of time steps in verification increases. Invariant inference [12, 23, 35] is another well-known approach for RNN verification. It constructs an invariant set of an RNN for verification without unrolling the network. The invariant inference approach is more scalable than the unrolling technique, but it may produce unknown verification results due to the accumulation of errors over time due to overapproximation.

In this paper, we extend the recent reachability method for verifying RNNs, using the notion of the *star set* representation. The star-set representation enables efficient computation of affine mapping and intersection operations and has been used for verification of FFNNs [3, 28, 29]. Its extension, ImageStar, can be used for verifying CNNs [26] and semantic segmentation networks (SSNs) [30] by efficient set propagation of average pooling and convolution layers. Reachability analysis of recurrent neural networks (RNNs) is challenging, as we need to deal with the "past information" stored in memory units. In RNN reachability, we have sequential or time-series input sets instead of sequential or time-series input points.

To solve this problem, we use star sets to derive exact and over-approximate reachability algorithms for RNNs with ReLU activation functions (piecewise activation functions in general) without unrolling. The key idea is that the dependencies between the current hidden states and previous hidden states are directly encoded as a set of constraints of the constructed reachable set (invariant) via the Minkowski sum operation of star sets. Therefore, the method can avoid the unrolling process. In exact reachability analysis, the number of output sets increases exponentially, making the verification of large RNNs a challenge. Therefore, we also present an over-approximation method that alleviates the scalability issue while still providing formal guarantees. We implement the algorithms in NNV [32], a verification tool for deep neural networks and learning-enabled cyber-physical systems. We evaluate the

HSCC '23, May 9–12, 2023, San Antonio, TX, USA
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0033-0/23/05…$15.00
https://doi.org/10.1145/3575870.3587128

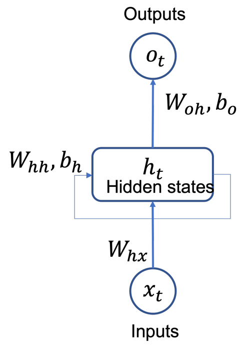

Figure 1: A recurrent neural network.

method in comparison with state-of-the-art methods RnnVerify and RNSVerify on a set of benchmarks. Experimental results show that our exact verification method is 10× faster than RNSVerify.

The main contributions of this paper are summarized as follows.

- A novel verification approach for recurrent neural networks.
- Exact and over-approximate reachability algorithms.
- Algorithms developed and included in the NNV software package with a
- A thorough evaluation and comparison with state-of-the-art techniques on a set of benchmarks.

2 BACKGROUND

2.1 Recurrent Neural Networks

A recurrent neural network, depicted in Figure 1, receives time-series (sequential) data x_t as inputs to compute and store the hidden states h_t that are used to obtain the outputs o_t. Mathematically, the hidden states and the output at the time step t are defined as:

$$\begin{aligned} h_t &= f_h(W_{hh}h_{t-1} + W_{hx}x_t + b_h), \\ o_t &= f_o(W_{oh}h_t + b_o), \end{aligned} \tag{1}$$

Where W_{hh}, W_{hx}, and W_{oh} are the hidden-state-to-hidden-state, input-to-hidden-state, hidden-state-to-output mapping matrices respectively; b_h and b_o are the hidden and output bias vectors; f_h and f_o are the activation functions. In this paper, we are interested in the Rectified Linear Unit (ReLU) activation function which is commonly used for training neural networks. One can see that the outputs o_t are computed from the hidden states h_t whose values depend on the previous hidden state h_{t-1}. Therefore, unlike FFNNs or CNNs, in an RNN, the previous inputs influence the current hidden states and outputs. We note that, the RNN shown in Figure 1 can be combined with an FFNN or an CNN to form a more complex network for a specific task, e.g., classification or prediction. Formally, an RNN forward computation is defined as follows.

*Definition 2.1 (**RNN Forward Computation**).* Given an input sequence $x = [x_1, x_2, \ldots, x_T]$, where x_i is the input vector at step i, the evaluation of the RNN on the input sequence x is the process of computing the sequence of hidden states $h_1, h_2, \ldots, h_T$ and the sequence of outputs $o = [o_1, o_2, \ldots, o_T]$. The initial hidden state h_o is initialized by users and is usually set as a zero vector. The evaluation of the RNN is done recursively using Equation 1.

An adversarial entity may slightly perturb the input of an RNN to generate an undesirable output.

*Definition 2.2 (**Attack on Inputs of an RNN**).* Given an input sequence $x = [x_1, x_2, \ldots, x_T]$, an ϵ-bounded attack $\mathcal{A}_\epsilon(x)$ alters the inputs in the sequence within an epsilon ϵ, i.e., $\mathcal{A}_\epsilon(x) : x_i \rightarrow x'_i, |x'_i - x_i| \leq \epsilon$. The attack creates a sequence of input sets $X = [X_1, X_2, \ldots, X_T]$ in which $X_i = \{x'_i | x_i - \epsilon \leq x'_i \leq x_i + \epsilon\}$.

To evaluate whether such an attack is feasible, we may conduct reachability analysis and evaluate the robustness of the RNN.

*Definition 2.3 (**Reachability of an RNN**).* Given a sequence of input sets $X = [X_1, X_2, \ldots, X_T]$, and an RNN $\mathcal{N}$, reachability analysis is the process of computing the corresponding sequence of outputs set $O = [O_1, O_2, \ldots, O_T]$ in which $O_i = \mathcal{N}(I_i), i = 1, 2, \ldots, T$.

Problem 1 (**Safety Verification of an RNN**). *Given an RNN $\mathcal{N}$, an input sequence $x = [x_1, x_2, \ldots, x_T]$, a set of linear constraints on the output sequence $P_o = P_{o_1} \wedge P_{o_2} \cdots \wedge P_{o_T}$ in which $P_{o_i} = \{o_i | C_i o_i \leq d_i\}$ representing the "unsafe region" that the output o_i should not reach (or shortly, $o_i \nvDash P_{o_i}$), and an attack $\mathcal{A}_\epsilon(x)$, determine whether thethe network is robust under the attack if for all $x'_i \in I_i = \{x'_i | x_i - \epsilon \leq x'_i \leq x_i + \epsilon\}, i = 1, 2, \ldots, T$, the corresponding output sequence $o' = [o'_1, o'_2, \ldots, o'_T]$ still does not satisfy the constraint P_o, i.e., o'_i does not satisfy $P_{o_i} : C_i o_i \leq d_i$ (or shortly, $o'_i \nvDash P_{o_i}$).*

Problem 2 (**Safety Verification of an RNN**). *Given a RNN $\mathcal{N}$, an input sequence $x = [x_1, x_2, \ldots, x_T]$, and an ϵ-bounded attack $\mathcal{A}_\epsilon(x)$, the robustness verification is to check whether the network is robust under the attack.*

We note that the robustness of an RNN is defined on a single input sequence $x = [x_1, x_2, \ldots, x_T]$. To evaluate the robustness of the network, we will compute its *average robustness* on a collection of N input sequences $C_x = \{x^1, x^2, \ldots, x^N\}$ in which $x^j = [x^j_1, x^j_2, \ldots, x^j_T]$. For example, if we can prove that the network is robust for 20 cases in a collection of 25 input sequences, then the average robustness of the network is $20/25 = 0.8$.

This paper focuses on verifying the robustness of a RNN using reachability analysis. The input sets created by the attack I_i, represented using star sets (defined in the following), are propagated through the network to compute the corresponding *reachable sets of the outputs* O_i. The reachable sets containing all possible outputs under the attack are used to verify the robustness of the network.

2.2 Star Set Representation

Definition 2.4 (Generalized Star Set [2, 6, 28]). A generalized star set (or simply star) Θ is a tuple $\langle c, V, P, l, u \rangle$ where $c \in \mathbb{R}^n$ is the center, $V = \{v_1, v_2, \cdots, v_m\}$ is a set of m vectors in $\mathbb{R}^n$ called basis vectors, $P : \mathbb{R}^m \rightarrow \{\top, \bot\}$ is a predicate, l and u are the lower-bound and upper-bound vectors of the predicate variables. The basis vectors are arranged to form the star's $n \times m$ basis matrix. The set of states represented by the star is given as:

$$\begin{aligned} [\![\Theta]\!] = \{x \mid x = c + \Sigma_{i=1}^m (\alpha_i v_i),\ P(\alpha_1, \cdots, \alpha_m) = \top, \\ l[i] \leq \alpha_i \leq u[i]\}. \end{aligned} \tag{2}$$

For the sake of brevity, we refer to both the tuple Θ and the set of states $[\![\Theta]\!]$ as Θ. In this work, we restrict the predicates to be a conjunction of linear constraints, $P(\alpha) \triangleq C\alpha \leq d$ where, for p

linear constraints, $C \in \mathbb{R}^{p\times m}$, α is the vector of m-variables, i.e., $\alpha = [\alpha_1, \cdots, \alpha_m]^T$, and $d \in \mathbb{R}^{p\times 1}$. A star is an empty set if and only if $P(\alpha)$ is empty.

Proposition 2.5. *Any bounded convex polyhedron* $\mathcal{P} \triangleq \{x \mid Cx \le d, x \in \mathbb{R}^n\}$ *can be represented as a star.*

Proposition 2.6. *[Affine Mapping of a Star] Given a star set* $\Theta = \langle c, V, P, l, u\rangle$, *an affine mapping of the star* Θ *with the affine mapping matrix* W *and offset vector* b *defined by* $\bar{\Theta} = \{y \mid y = Wx+b,\ x \in \Theta\}$ *is another star with the following characteristics:* $\bar{\Theta} = \langle \bar{c}, \bar{V}, \bar{P}, \bar{l}, \bar{u}\rangle$, $\bar{c} = Wc + b$, $\bar{v} = \{Wv_1, Wv_2, \cdots, Wv_m\}$, $\bar{P} \equiv P, \bar{l} \equiv l, \bar{u} \equiv u$.

Proposition 2.7 (Star and Half-space Intersection). *The intersection of a star* $\Theta \triangleq \langle c, V, P, l, u\rangle$ *and a half-space* $\mathcal{H} \triangleq \{x \mid Hx \le g\}$ *is another star with the following characteristics.*

$$\bar{\Theta} = \Theta \cap \mathcal{H} = \langle \bar{c}, \bar{V}, \bar{P}, \bar{l}, \bar{u}\rangle,\ \bar{c} = c,\ \bar{V} = V,\ \bar{P} = P \wedge P',$$
$$P'(\alpha) \triangleq (H \times V_m)\alpha \le g - H \times c, V_m = [v_1\ v_2 \cdots v_m],$$
$$\bar{l} = l,\ \bar{u} = u.$$

Proposition 2.8. *[Minkowski sum of two stars] Given two stars* $\Theta_1 = \langle c_1, V_1, P_1, l_1, u_1\rangle$ *and* $\Theta_2 = \langle c_2, V_2, P_2, l_2, u_2\rangle$ *with the same dimensions, the Minkowski sum of the two stars is another star set* $\Theta = \Theta_1 \oplus \Theta_2 = \langle c, V, P, l, u\rangle$, *where* $c = c_1 + c_2$, $V = [V_1\ V_2]$, $P = P_1 \wedge P_2$, $l = [l_1\ l_2]^T$, *and* $u = [u_1\ u_2]^T$.

The advantages in computing affine mapping, intersection, and Minkowski addition make star set an efficient approach to the computation of reachable sets of an RNN given a sequence of star input sets. Star input sets will be defined in detail in the next section.

Proposition 2.9. *[Optimized range of a state] Given a star set* $\Theta = \langle c, V, P, l, u\rangle$, *the range of the* i^{th} *state* $x[i]$ *of the star set can be found by solving the following linear programming optimization problems:*

$$x[i]_{min(max)} = min(max)(c[i] + \Sigma_{j=1}^{m} v_j[i]\alpha_j),$$
$$s.t.\ P(\alpha) \triangleq C\alpha \le d,\ l \le \alpha \le u.$$

Proposition 2.10. *[Estimated range of a state] Given a star set* $\Theta = \langle c, V, P, l, u\rangle$, *the range of the state vector* x *of the star set can be estimated quickly without solving the linear programming optimization problems by using only the lower bound and upper bound vectors of the predicate variables.*

$$l_{est} \le x = c + V\alpha = c + max(0, V)\alpha + min(0, V)\alpha \le u_{est},$$
$$l_{est} = c + max(0, V)l + min(0, V)u,$$
$$u_{est} = c + max(0, V)u + min(0, V)l.$$

The optimized (estimated) ranges of all states in a star set are used to construct an over-approximate reachable set of an RNN using approximate (relaxed) reachability, which is studied in the next section.

3 REACHABILITY ANALYSIS OF AN RNN

This section focuses on investigating the exact and approximate reachability of a ReLU RNN. The exact reachability algorithm is used to compute the exact reachable set of the network, while the approximate reachability algorithm provides a tight overapproximation of the exact reachable sets.

3.1 Exact Reachability

From Equation 1, we can see that the reachable set of the output O_t is obtained by applying the ReLU activation function f_o on the affine map of the reachable set of the hidden states, i.e., $O_t = ReLU(\bar{H}_t) = ReLU(W_{oh}H_t + b_o)$. Therefore, the key is to compute the reachable set of the hidden states H_t, which depends on their previous reachable set H_{t-1} and the current input sets X_t:

$$H_t = ReLU((W_{hh}H_{t-1} + b_h) \oplus W_{hx}X_t),$$

in which $\oplus$ is the Minkowski sum operation.

Computing the first output set. Without loss of generality, we assume that the initial hidden states (h_0) are set to zeros and the input set X_t is a star set $X_t = \langle c_t^x, V_t^x, P_t^x, l_t^x, u_t^x\rangle$. Using Proposition 2.6, we have:

$$\tilde{H}_1 = W_{hx}X_1 + b_h = \langle \tilde{c}_1, \tilde{V}_1, \tilde{P}_1, \tilde{l}_1, \tilde{u}_1\rangle,$$
$$\tilde{c}_1 = W_{hx}c_1^x + b_h,\ \tilde{V}_1 = W_{hx}V_1^x,\ \tilde{P}_1 \equiv P_1^x,\ \tilde{l}_1 \equiv l_1^x,\ \tilde{u}_1 \equiv u_1^x.$$

The reachable set of the first hidden states is $H_1 = ReLU(\tilde{H}_1)$, which can be computed exactly by a sequence of *stepReLU* operations [27, 28] as follows.

$$H_1 = ReLU_n(ReLU_{n-1}(\ldots(ReLU_1(\tilde{H}_1)))),$$

where n is the dimension of the star set $\tilde{H}_1$ and $ReLU_i$ is the stepReLU operation applied to the i^{th} state of the star set. It is known that each stepReLU operation may split one *intermediate* star set into two new star sets. Therefore, the exact reachable set of the first hidden states H_1 may consist of multiple star sets.

$$H_1 = [H_1^1, H_1^2, \ldots, H_1^m], H_1^i = \langle c_1^i, V_1^i, P_1^i, l_1^i, u_1^i\rangle$$

From the exact reachable set of the first hidden states H_1, we perform affine mapping operation and then apply ReLU activation function on the obtained star sets to compute the corresponding output set O_1 as follows:

$$\bar{H}_1^i = W_{oh}H_1^i + b_o,\ i = 1, 2, \ldots, m$$
$$O_1^i = ReLU_p(ReLU_{p-1}(\ldots(ReLU_1(\bar{H}_1)))),$$
$$O_1 = [O_1^1, O_1^2, \ldots, O_1^m],$$

where p is the dimension of the output state vector o_1.

Computing the $(i+1)^{th}$ output set. Assume that at the step i, the exact reachable set of the hidden states H_i contains m star sets, i.e., $H_i = [H_i^1, H_i^2, \ldots, H_i^m], H_i^j = \langle c_i^j, V_i^j, P_i^j, l_i^j, u_i^j\rangle$, and the input set is $X_i = \langle c_i^x, V_i^x, P_i^x, l_i^x, u_i^x\rangle$. The reachable set of the $(i+1)^{th}$ hidden states H_{i+1} is computed as follows.

$$\tilde{H}_{i+1}^j = (W_{hh}H_i^j + b_h) \oplus W_{hx}X_i = \langle \tilde{c}_i, \tilde{V}_i, \tilde{P}_i, \tilde{l}_i, \tilde{u}_i\rangle,$$
$$\tilde{c}_i = W_{hh}c_i^j + b_h + W_{hx}c_i^x,\ \tilde{V}_i = [W_{hh}V_i\ W_{hx}V_i^x],$$
$$\tilde{P}_i = P_i^j \wedge P_i^x, \tilde{l}_i = [l_i^j\ l_i^x]^T, \tilde{u}_i = [u_i^j\ u_i^x]^T,$$
$$H_{i+1}^j = ReLU_n(ReLU_{n-1}(\ldots(ReLU_1(\tilde{H}_i^j)))),$$
$$H_{i+1}^j = [H_{i+1}^{j,1}, H_{i+1}^{j,2}, \ldots, H_{i+1}^{j,M_j}],$$
$$H_{i+1} = [H_{i+1}^1, H_{i+1}^2, \ldots, H_{i+1}^m].$$

We can see that, for individual hidden states star set H_i^j at step i, the exact reachability scheme obtains a new hidden states reachable set H^j_{i+1} consisting of multiple star sets $H^{j,1}_{i+1}, H^{j,2}_{i+1}, \ldots, H^{j,M_j}_{i+1}$ due to splitting in stepReLU operations.

For each individual hidden states star set $H^{j,k}_{i+1}$ at step $i+1$, we compute the corresponding output set O_{i+1} as in the first step.

$$\bar{H}^{j,k}_{i+1} = W_{oh}H^{j,k}_{i+1} + b_o,\ j = 1, 2, \ldots, m,\ k = 1, 2, \ldots, M_j,$$
$$O^{j,k}_{i+1} = ReLU_p(ReLU_{p-1}(\ldots(ReLU_1(\bar{H}^{j,k}_{i+1})))),$$
$$O^{j,k}_{i+1} = [O^{j,k,1}_{i+1}, O^{j,k,2}_{i+1}, \ldots, O^{j,k,N}_{i+1}].$$

Exact Reachability. Algorithm 3.1 shows the process of computing the reachable set of an RNN $\mathcal{N} = \langle W_{hh}, b_h, W_{hx}, W_{oh}, b_o\rangle$ given a sequence of T star input sets $X = [X_1, X_2, \ldots, X_T]$. The algorithm computes the exact reachable set of hidden states H_i at each step (lines 3 to 9) before obtaining the exact reachable set of the output states (lines 10 to 14). The exact reachable set of the hidden states is computed based on their previous reachable set and the current input set (lines 7 to 9). We note that the ReLU operation in the algorithm is performed via a sequence of stepReLU operations (lines 4, 9, and 14) to construct the exact reachable set of the network. The sub-procedure for the stepReLU operation at the neuron i, i.e., $ReLU_i(\cdot)$ is presented from lines 16 to 27.

Remark 1. *The exact reachability algorithm directly computes the reachable set of the hidden states (lines 2-10 in Algorithm 3.1) without unrolling the network as the RNSVerify approach [1]. By taking advantage of the Minkowski sum of star sets, a new star representing the current hidden state reachable set can be constructed directly and efficiently from the previous hidden state reachable set and the current input set (line 8 in Algorithm 3.1).*

Implementation Highlight. In the implementation of the exact reachability algorithm, we do not need to find the exact range of the input to each neuron (Proposition 2.9), i.e., l_i and u_i (line 19), which is computationally expensive. Instead, we only need to quickly determine if a split occurs at a specific neuron using its estimated range (Proposition 2.10) or using zonotope prefilter techniques [3, 29] to minimize the number of LPs solved in the exact analysis.

Complexity of the Exact Reachability.

Lemma 3.1. *Given an RNN network $\mathcal{N}$ with an n-dimensional hidden states space, and p-dimensional output space, the complexity of the number of star sets in the exact reachability of the network at step T in the worst-case is $O(2^{Tn+p})$. The complexity of the number of constraints in the predicate of a star in the reachable set of the output states is $O(T(n + nc_{max}) + p)$, where nc_{max} is the maximum number of constraints in the input sets.*

Proof. At the first step, the number of star sets in the reachable set of the hidden states H_1 is at most 2^n (line 4) as each stepReLU operation may split a star set into two new star sets. Therefore, the number of star sets in the reachable set of the output O_1 (line 14) is at most $2^n 2^p = 2^{n+p}$. At the second step, the number of star sets in the reachable set of the second hidden states H_2 is at most $2^n 2^n = 2^{2n}$. Consequently, the number of star sets in the reachable set of the output O_2 is at most $2^{2n}2^p = 2^{2n+p}$. It is easy to generalize that at step T, the number of the star sets in the reachable set of the hidden states H_T is at most 2^{Tn}, and the corresponding number of star sets in the output reachable set is at most 2^{Tn+p}.

Algorithm 3.1 Exact Reachability of an RNN.

```
Input: $\mathcal{N} = \langle W_{hh}, b_h, W_{hx}, W_{oh}, b_o\rangle$, $X = [X_1, X_2, \ldots, X_T]$
Output: $O = [O_1, O_2, \ldots, O_T]$
1:  procedure $O$ = ExactReach($\mathcal{N}, X$)
2:      for $i = 1 : T$ do
3:          if i=1 then $\tilde{H}_i = W_{hx}X_i + b_h$
4:              $H_i = ReLU_n(ReLU_{n-1}(\ldots(ReLU_1(\tilde{H}_i))))$
5:          else
6:              $m = length(H_{i-1})$
7:              for $j = 1 : m$ do
8:                  $\tilde{H}^j_i = (W_{hh}H^j_{i-1} + b_h) \oplus W_{hx}X_i$
9:                  $H^j_i = ReLU_n(ReLU_{n-1}(\ldots(ReLU_1(\tilde{H}^j_i))))$
10:                 $H_i \leftarrow H^j_i$
11:         $M = length(H_i)$
12:         for $k = 1 : M$ do
13:             $\bar{H}^k_i = W_{oh}H^k_i + b_o$
14:             $O^k_i = ReLU_p(ReLU_{p-1}(\ldots(ReLU_1((\bar{H}^k_i))))$
15:             $O_i \leftarrow O^k_i$
16: procedure $\tilde{R} = ReLU_i(\tilde{I})$
17:     $\tilde{R} = \emptyset$, $\tilde{I} = [\tilde{\Theta}_1\ \cdots\ \tilde{\Theta}_k]$
18:     for $j = 1 : k$ do
19:         $[l_i, u_i] = \tilde{\Theta}_j.getRange(i)$   ▷ range of the input to the $i^{th}$ neuron
20:         $R_1 = \emptyset$, $M = [e_1\ e_2\ \cdots e_{i-1}\ 0\ e_{i+1}\ \cdots\ e_n]$
21:         if $l_i \geq 0$ then $R_1 = \tilde{\Theta}_j = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}_j, \tilde{l}_j, \tilde{u}_j\rangle$
22:         if $u_i \leq 0$ then $R_1 = M * \tilde{\Theta}_j = \langle M\tilde{c}_j, M\tilde{V}_j, \tilde{P}_j, \tilde{l}_j, \tilde{u}_j\rangle$
23:         if $l_i < 0\ \&\ u_i > 0$ then
24:             $\tilde{\Theta}'_j = \tilde{\Theta}_j \wedge x[i] \geq 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}'_j, \tilde{l}_j, \tilde{u}_j\rangle$
25:             $\tilde{\Theta}''_j = \tilde{\Theta}_j \wedge x[i] < 0 = \langle \tilde{c}_j, \tilde{V}_j, \tilde{P}''_j, \tilde{l}_j, \tilde{u}_j\rangle$
26:             $R_1 = \tilde{\Theta}'_j \cup M * \tilde{\Theta}''_j$
27:         $\tilde{R} = \tilde{R} \cup R_1$
```

At each stepReLU operation, a star set may be split into two new stars (line 23 to 26). Each new star set has one more constraint, i.e., $x[i] \leq 0$ or $x[i] \leq 0$ (line 24 and 25). Therefore, at the first step, a star set in the reachable set of the hidden states H_1 has at most $n + nc_1$ constraints where nc_1 is the number of contraints in the predicate of the input set X_1 (nc_1 is the number of columns of the predicate matrix $P^x_1.C$). Consequently, a star set in the reachable set of the first output O_1 has at most $n + nc_1 + p$ constraints. One can see that, at step T, a star set in the reachable set of the hidden states H_T has at most $n + nc_1 + n + nc_2 + \ldots n + nc_T = Tn + \Sigma^T_1 nc_i$. Assume that $nc_i \leq nc_{max}, i = 1, \ldots, T$, then the complexity of the number of constraints in the predicate of a star in the reachable set of the output is $T(n + nc_{max}) + p$. □

Remark 2. *The computational complexity of the exact reachability algorithm indicates that, as the size of the network increases, the potential number of star sets may increase exponentially. The number of constraints of a star set in the reachable set is useful for estimating the theoretical scalability of our verification approach using state-of-the-art LP solvers. We expand this analysis in the next section.*

3.2 Overapproximation Method

To overcome the explosion in the number of hidden-state and output star sets in the exact reachability, i.e., line 4, 9, and 14 in Algorithm 3.1, we overapproximate the stepReLU operation $ReLU_i$ by a triangle over-approximation rule. Let l_i and u_i be the lower bound and upper

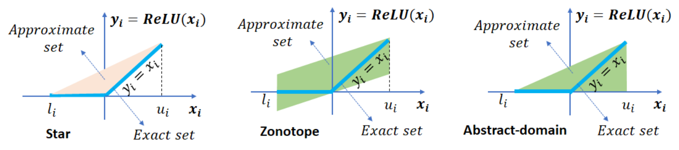

Figure 2: The exact output y_i of the ReLU activation function on a neuron x_i where splitting occurs, i.e., $l_i \leq x_i \leq u_i$, $l_i < 0$, $u_i > 0$ can be overapproximated by a triangle (our method), or a zonotope [24] or an abstract-domain [25]. This overapproximation domain can be captured by a single new variable α_{new} with associated constraints. The triangle overapproximation is the tightest one.

bound of the input $x[i]$ (a star set) to the i^{th} neuron in a n-neurons (hidden/output) layer (after an affine mapping operation). If $l_i < 0$ and $u_i > 0$, applying ReLu activation function on this neuron causes a splitting which results in two separate segments shown in Figure 2 (in blue color). We can overapproximate these two segments by a single set using a triangle (our method) or a zonotope [24] or an abstract-domain [25]. We represent the triangle set by a new predicate variable α_{new}, i.e., $y[i] = ReLU(x[i]) = \alpha_{new}$, with three associating constraints in the following:

$$\alpha_{new} \leq 0,\ \alpha_{new} \geq x[i],\ \alpha_{new} \leq \frac{u_i}{u_i - l_i}(x[i] - l_i).$$

When applying the triangle over-approximation rule for all splitting neurons in the layer, we can construct a single star set that is an over-approximation of the exact reachable set (containing multiple star sets). The overapproximate set has at most n new predicate variables and $3n$ new constraints.

The lower bound l and upper bound u vectors of the input x to the layer are required to construct an overapproximate reachable set. These bounds can be obtained by solving $2n$ LP optimization problems (Proposition 2.9) which is computationally expensive. To reduce the computation time, we combine the optimized ranges (Proposition 2.9) and estimated ranges (Proposition 2.10) of neurons in constructing the approximate reachable set. The idea is to choose neurons at which the overapproximation causes potentially the largest errors, i.e., the neurons with the largest triangle areas, to optimize their ranges and then combine with estimated ranges from other neurons with smaller overapproximation errors (small triangle areas) to construct a *relaxed* approximate reachable set. The relaxed reachability method requires a *relaxation factor (RF)*, scaled from 0% to 100% as an input [30]. When $RF = 0$, there is no relaxation in the approximate reachability. When $RF = 100\%$, we use the estimated ranges of all neurons to construct the reachable set. The larger the RF, the smaller the required computation time. When we fully relax the approximate reachability, i.e., $RF = 100\%$, the verification process becomes very fast and can even be used in a real-time setting. We study the effect of various relaxation factors in detail in the evaluation section.

Approximate Reachability Algorithm. Algorithm 3.2 describes the relaxed approximate reachability of an RNN, which constructs the relaxed overapproximation reachable sets of the hidden states H_i (line 3, 4, and 5) and the output O_i (line 6 and 7) using the $ReLU_{relax}$ procedure (line 8 to 19). The $ReLU_{relax}$ procedure estimates the ranges of all states in a star set quickly using Proposition 2.10 (line 9). Using the estimated ranges, it finds all indexes of the states at which overapproximations are needed (line 10). It then ranks these indexes using their estimated triangle overapproximation areas, i.e., $S_i = 0.5u_i(u_i - l_i)$ (line 11). From the ranked indexes, we choose $RF \times N$ indexes with the highest ranks, i.e., the largest estimated overapproximation areas, to optimize their ranges by solving LP optimizations using Proposition 2.9 (line 13 and 14). Finally, it combines the optimized ranges and estimated ranges (lines 15 and 16) before using relaxed stepReLU $ReLU_j^{rl}$ to construct the reachable set (lines 20 to 36). Note that the relaxed stepReLU uses mixed ranges from the previous step instead of optimized ranges for all neurons.

Algorithm 3.2 Relaxed Reachability of an RNN.

```
Input: $\mathcal{N} = \langle W_{hh}, b_h, W_{hx}, W_{oh}, b_o \rangle$, $X = [X_1, X_2, \ldots, X_T]$, $RF$
Output: $O = [O_1, O_2, \ldots, O_T]$
 1: procedure $O$ = RelaxReach($\mathcal{N}, X, RF$)
 2:     for $i = 1 : T$ do
 3:         if i=1 then $\tilde{H}_i = W_{hx}X_i + b_h$
 4:         else $\tilde{H}_i = (W_{hh}H_{i-1} + b_h) \oplus W_{hx}X_i$
 5:         $H_i = ReLU_{relax}(\tilde{H}_i, RF)$
 6:         $\bar{H}_i = W_{oh}H_i + b_o$
 7:         $O_i = ReLU_{relax}(\bar{H}_i, RF)$
 8: procedure $R = ReLU_{relax}(\Theta, RF)$
 9:     $[l, u] = \Theta.estimateRanges$  ▹ estimate ranges of all states
10:     $ids = find(j, l[j] < 0, u[j] > 0)$
11:     $ranked\_ids = rank(id \in ids, u[id] \times (u[id] - l[id]))$
12:     $N = length(ranked\_ids)$
13:     $opt\_ids = ranked\_ids[1 : RF \times N]$
14:     $[l_{opt}, u_{opt}] = \Theta.getRanges(opt\_ids)$
15:     $l_{mix} = l(l[opt\_ids] \leftarrow l_{opt})$
16:     $u_{mix} = u(u[opt\_ids] \leftarrow u_{opt})$
17:     $R = \Theta$
18:     for $j = 1 : n$ do
19:         $R = ReLU_j^{rl}(R, l_{mix}, u_{mix})$
20:     procedure $\tilde{R} = ReLU_j^{rl}(\tilde{\Theta}, l_{mix}, u_{mix})$
21:         $M = [e_1\ e_2\ \cdots e_{j-1}\ 0\ e_{j+1}\ \cdots\ e_n]$
22:         if $l_{mix}[j] \geq 0$ then $\tilde{R} = \tilde{\Theta} = \langle \tilde{c}, \tilde{V}, \tilde{P}, \tilde{l}, \tilde{u} \rangle$
23:         if $u_{mix}[j] \leq 0$ then $\tilde{R} = M * \tilde{\Theta} = \langle M\tilde{c}, M\tilde{V}, \tilde{P}, \tilde{l}, \tilde{u} \rangle$
24:         if $l_{mix}[j] < 0$ & $u_{mix}[j] > 0$ then
25:             $\tilde{P}(\alpha) \triangleq \tilde{C}\alpha \leq \tilde{d}$, $\alpha = [\alpha_1, \alpha_2, \cdots, \alpha_m]^T$
26:             $\alpha' = [\alpha_1, \cdots, \alpha_m, \alpha_{m+1}]^T$  ▹ new variable $\alpha_{m+1}$
27:             $C_1 = [0\ 0\ \cdots\ 0\ \text{-}1]$, $d_1 = 0$  ▹ $\alpha_{m+1} \geq 0$
28:             $C_2 = [\tilde{V}(j,:)\ \text{-}1]$, $d_2 = -\tilde{c}[j]$  ▹ $\alpha_{m+1} \geq x[j]$
29:             $\gamma = \frac{u_{mix}[j]}{u_{mix}[j] - l_{mix}[j]}$  ▹ $\alpha_{m+1} \leq \gamma(x[j] - l_{mix}[j])$
30:             $C_3 = [-\gamma\tilde{V}(j,:)\ \ 1]$, $d_3 = \gamma l_{mix}[j](1 - \tilde{c}[j])$
31:             $C_0 = [\tilde{C}\ 0_{m\times1}]$, $d_0 = \tilde{d}$
32:             $C' = [C_0; C_1; C_2; C_3]$, $d' = [d_0; d_1; d_2; d_3]$
33:             $P'(\alpha') \triangleq C'\alpha' \leq d'$
34:             $c' = M\tilde{c}$, $V' = M\tilde{V}$, $V' = [V'\ e_j]$
35:             $l' = [\tilde{l}; 0]$, $u' = [\tilde{u}; u_{mix}[j]]$
36:             $\tilde{R} = \langle c', V', P', l', u' \rangle$
```

Implementation Highlight. The relaxed stepReLU operations (line 18 and 19) in Algorithm 3.2 is written in a *for loop* for easy understanding. In our implementation, after computing the mixed ranges, we perform the relaxed stepReLU operations on multiple neurons simultaneously to reduce the computation time.

4 VERIFICATION OF RNNS

The goal of this research is to verify the robustness of an RNN under an adversarial attack, i.e., solve Problem 1. To achieve that, we first represent the adversarial inputs caused by the attack $\mathcal{A}_\epsilon(x)$ as a sequence of stars $X = [X_1, X_2, \ldots, X_T]$ in which $X_i = \{x'_i | x_i - \epsilon \leq x'_i \leq x_i + \epsilon = \langle c^x_i, V^x_i, P^x_i, l^x_i, u^x_i \rangle\}$. Using the reachability methods in the previous section, we then compute the reachable set of outputs of the network $O = [O_1, O_2, \ldots, O_T]$ corresponding to the sequence of stars input set X. Finally, we check if there is the intersection between each star output set O_i and its corresponding *unsafe property* $P_{o_i} \triangleq C_i o_i \leq d_i$. If the output set O_i does not reach its unsafe property, then the network is *ROBUST* under the attack. Otherwise, depending on the reachability methods used in verification, the result can be *NOT ROBUST* or *UNKNOWN*.

Verification Algorithm. Algorithm 4.3 describes our verification approach for an RNN using star reachability. The Algorithm receives the network $\mathcal{N}$, the input sequence $x = [x_1, x_2, \ldots, x_T]$, the unsafe property $P_o = P_{o_1} \wedge \cdots \wedge P_{o_T}$, the input disturbance bound ϵ, the reachability method *method* and the relaxation factor *RF* (for relaxed reachability) as inputs. It outputs the verification result and counterexample (if exact reachability is used). From the input sequence x and the disturbance bound ϵ, we construct the sequence of input sets X (line 2 to 4). We then compute the reachable set of the outputs given the sequence input sets and the reachability method (line 5 to 8). Finally, we verify the robustness of the network using the output reachable set O and its corresponding unsafe property P_o by initializing the result as *ROBUST* (line 9). If the output reachable set O intersect with the unsafe property P_o and the relaxed reachability algorithm is used, then the verification result is *UNKNOWN* (line 13). Similarly, but if the exact reachability algorithm is used, the verification result is *NOT_ROBUST* (line 15) and a set of counter examples is constructed (line 16).

Algorithm 4.3 Robustness Verification of an RNN.

```
Input: $\mathcal{N}, x = [x_1, x_2, \ldots, x_T], P_o = P_{o_1} \wedge \cdots \wedge P_{o_T}, \epsilon, method, RF$
Output: $ROBUST, NOT\_ROBUST, UNKNOWN, CEx$
 1: procedure  $result$ = Verify($\mathcal{N}, x, \epsilon, method, RF$)
 2:     for $i = 1 : T$ do  ▷ construct input sets
 3:         $X_i = \{x'_i | x_i - \epsilon \leq x'_i \leq x_i + \epsilon\} = \langle c^x_i, V^x_i, P^x_i, l^x_i, u^x_i \rangle$
 4:         $X \leftarrow X_i$
 5:     if $method$ = "$exactstar$" then
 6:         $O = ExactReach(\mathcal{N}, X)$  ▷ Algorithm 3.1
 7:     else if $method$ = "$relaxstar$" then
 8:         $O = RelaxReach(\mathcal{N}, X, RF)$  ▷ Algorithm 3.2
 9:     $result = ROBUST, CEx = \emptyset$
10:     for $i = 1 : T$ do
11:         if $O_i \cap P_{o_i} \neq \emptyset$ then
12:             if $method$ = "$relaxstar$" then
13:                 $result = UNKNOWN$
14:             else if $method$ = "$exactstar$" then
15:                 $result = NOT\_ROBUST$
16:                 $CEx = GetCounterExample(O, P_o, i)$
17:             break
```

Counterexample Construction. It is important to emphasize that, using the exact reachability algorithm, we can construct a *complete set of counterexamples* containing all input sequences $CEx = \{x^1, x^2, \ldots, x^\infty\}$, $x^j = [x^j_1, x^j_2, \ldots, x^j_T]$ that make the network violates its property, i.e., the output sequences $CO = \{o^1, o^2, \ldots, o^\infty\}$, $o^j = [o^j_1, o^j_2, \ldots, o^j_T]$ reach the unsafe region defined by P_o. The set of counterexamples is constructed by applying the lemma.

Lemma 4.1. ***[Counterexample input sets.]*** *Given a sequence of input sets $X = [X_1, X_2, \ldots, X_T]$, $X_i = \langle c^x_i, V^x_i, P^x_i, l^x_i, u^x_i \rangle \in \mathbb{R}^q$, and the exact output set at step i is $O_i = [O^1_i, O^2_i, \ldots, O^{N_i}_i]$, if the network violates its robustness property at step i, then the complete counter input sets containing all possible counter input sequences is $CE = [CE_j]$, $1 \leq j \leq N_i$ in which:*

$$O^j_i \cap P_{o_i} \triangleq C_i o_i \leq d_i = \langle c^{o_j}_i, V^{o_j}_i, P^{o_j}_i, l^{o_j}_i, u^{o_j}_i \rangle \neq \emptyset,$$
$$CE_j = [\tilde{X}^j_1, \tilde{X}^j_2, \ldots, \tilde{X}^j_k, \ldots, \tilde{X}^j_i],\ k \leq i$$
$$\tilde{X}^j_k = \langle c^x_k, [0_{q\times(k-1)q}\ V^x_k\ 0_{q\times(i-k)q}], P^{o_j}_i, l^{o_j}_i, u^{o_j}_i \rangle.$$

Proof. We remind that 1) the affine mapping operation (Prop. 2.6) does not change the predicate of a star set, 2) the intersection between a star set and a halfspace does not change the center and basis vectors of the star set and the new constraints are added to the new predicate (Proposition 2.7), 3) the exact ReLU operation does not increase the number of predicate variables but may add more constraints to them, and 4) the Minkowski sum increases the size of the basis vectors and the number of the predicate variables as wells as the number of constraints in the new predicate (Pro 2.8). Due to the Minkowski sum (line 8 in Algorithm 3.1), the number of predicate variables of the output reachable set at step i is $i \times q$. In the exact analysis, the predicate of the output reachable set O_i is built upon the predicates of the input sets $X_1, X_2, \ldots, X_i$ by adding more constraints via the ReLU operation (lines 4, 9, and 14 in Algorithm 3.1) and the Minkowski sum (line 8 in the algorithm). Therefore, if $P^{o_j}_i$ is the predicate of the unsafe output $U^j_i = O^j_i \cup P_{o_i}$, it contains all constraints of the predicate variables of all corresponding unsafe input sets $\tilde{X}^j_1, \tilde{X}^j_2, \ldots, \tilde{X}^j_k, \ldots, \tilde{X}^j_i$ in which $\tilde{X}^j_k$ is a subset of X_k. Note that $P^{o_j}_i$ contains the constraints of $i \times q$ predicate variables while the input set X_k is defined based on q predicate variables. Therefore, the unsafe input set $\tilde{X}^j_k (\subset X_k)$ is constructed based on $i \times q$ predicate varables. However, it does not depend on the first $(k-1)q$ and the later $(i-k)$ predicate variables. This means its basis vectors corresponding to these variables are zero vectors, i.e., $\tilde{V}^j_k = [0_{q\times(k-1)q}\ V^x_k\ 0_{q\times(i-k)q}]$. □

Soundness and Completeness. The soundness and completeness of our verification algorithm is described in the following lemma.

Lemma 4.2. *If the exact reachability algorithm is used, Algorithm 4.3 is sound and complete. Otherwise, it is sound.*

5 EVALUATION

Experiment Setup. Our approach is implemented in NNV, a tool for verification of deep neural networks and learning-enabled cyber-physical systems [32]. All codes used to produce the results in this paper are available for repeatability evaluation. We evaluate our method in comparison with RnnVerify [12] and RNSVerify [1] on the adversarial robustness verification for speaker recognition RNN benchmarks (exactly the same as RnnVerify does). The RnnVerify method implements the invariant inference approach to reduce the complexity of verifying RNNs. The RNSVerify method is based on the *unrolling* technique that transforms an RNN into an equivalent feedforward network which can be encoded as a big MILP to reason about the robustness. Similar to our exact verification approach, RNSVerify is sound and complete, but computationally expensive.

Benchmarks. We use 6 speaker recognition RNNs [12] trained on the VCTK dataset to compare our method with RnnVerify. Each network $N_{a,b}$ has an input layer of 40 dimensions, two hidden layers with $a = \{2, 4, 8\}$ and $b = \{0, 2, 4, 8\}$ memory units, followed by 5 fully connected layers with 32 ReLU nodes each, and an output layer with 20 ReLU nodes. The output nodes represent the possible speakers in which the one with the highest score is the classified output, i.e., the recognized speaker. We refer readers to [12] for further details of the networks. We also use a small network with one hidden layer (in RnnVerify) to evaluate our exact verification approach. Our experiment is done on a computer with the following configurations: Intel Core i7-10700 CPU @ 2.90GHz × 8 Processors, 63.7 GiB Memory, 64-bit Ubuntu 18.04.6 LTS OS.

Adversarial Robustness Verification. We use the same 25 fixed input points $x = \{x_1, x_2, \ldots, x_{25}\}$ generated randomly in RnnVerify for our comparison. These input points are 40-dimensional vectors and do not change over time, i.e., $x_i \in \mathbb{R}^{40}$, $x_i^1 = x_i^2, \ldots, = x_i^{T_{max}}$. For a network $\mathcal{N}_{a,b}$ and input x_i, the ground truth label corresponding to the highest score output of the network at step T_{max} is $l_{T_{max}} = \mathcal{N}_{a,b}(x_i^1, x_i^2, \ldots, x_i^{T_{max}})$. We want to verify that if we arbitrarily attack the input sequence $x_i^1, x_i^2, \ldots, x_i^{T_{max}}$ by bounded perturbation $\epsilon = 0.01$, then whether the output label at the step T_{max} is still the same, i.e., $l'_{T_{max}} = \mathcal{N}_{a,b}(x_i'^1, x_i'^2, \ldots, x_i'^{T_{max}}) = l_{T_{max}}$, where $\bigwedge_{t=1}^{T_{max}} \|x_i'^t - x_i^t\|_\infty \leq 0.01$.

5.1 Exact Verification

We evaluate our exact verification approach on a small RNN with one hidden layer to compare its results with RnnVerify and RNSVerify. We analyze the timing performance, the growth of the number of star sets, and counterexample constructions of our approach. The results of the exact reachability verification for the small RNN network are presented in Table 1.

Timing Performance. From Table 1, we can see that the verification time varies significantly with different input points. Input points $x_4, x_{10}, x_{21}, x_{22}$ and x_{24} dominate the verification time. When T_{max} increases, the verification time grows quickly in the exact analysis. Our exact verification scheme significantly outperforms the RNSVerify method, as shown in Figure 3. We can achieve ≈ 10× times faster than RNSVerify on average when $T_{max} = 20$. One can also see from the figure that, the exact verification scheme is slower than the approximate one and RnnVerify when T_{max} is large.

Number of Star Sets. From the table, one can see that for those input points that dominate the verification time, the number of star sets in the output reachable set grows significantly when we increase T_{max}. For example, for the input point x_{24}, the number of star output sets increases from 3 at $T_{max} = 5$ to 4181 at $T_{max} = 20$. For other input points, the number of star sets grows slowly (e.g., x_1, x_6) or does not grow (e.g., x_2, x_3) as T_{max} increases.

Counterexamples. In RnnVerify, the paper claims that for the small network, they can prove the robustness of the networks for all 25 input points for all T_{max} between 2 and 20. In our result, we have found many counterexamples to prove that the network is not robust. For example, at $T_{max} = 20$, using our exact verification scheme, we can prove that the network is not robust for 12 of 25 input points (see Table 1). Figure 4 shows the ranges of all outputs of the network for the first output set in 8 counter output sets for x_1 with $T_{max} = 20$. One can see that there is the case that the 10th output will be the largest one. Therefore, the network may result in the label 10 instead of 1 as expected. We note that the exact verification results are consistent with the ones obtained by our approximate verification scheme. For the cases that are not robust, our approximate verification scheme produces UNKNOWN as a result. We have tried the RnnVerify with different attack bound values, i.e., ϵ. We have experienced that even if we let $\epsilon = 1$, which is a significantly large bound, RnnVerify still proves that the network is robust for all 25 input points.

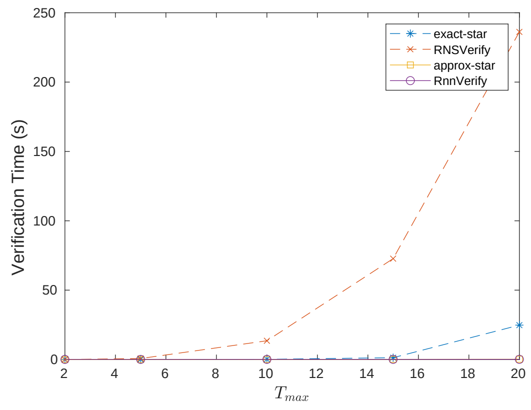

Figure 3: Average verification times of different approaches for the small RNN network. Our exact verification significantly outperforms RNSVerify.

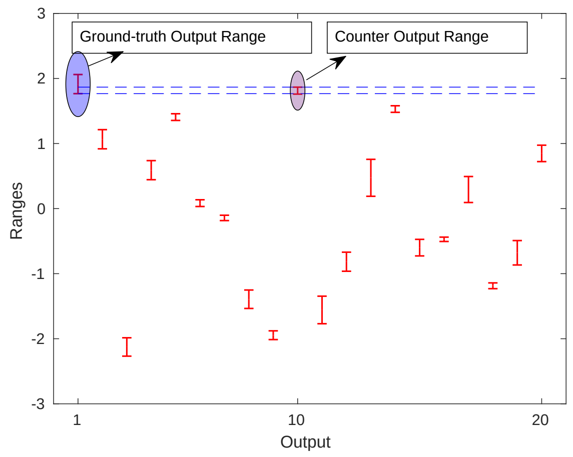

Figure 4: The figure shows the ranges of all outputs in the first output set in 8 counter output sets for x_1 with $T_{max} = 20$ (Table 1). There is the case that the output label is 10 while it is expected to be 1.

x	Tmax = 5				Tmax = 10				Tmax = 15				Tmax = 20			
	RS	VT	N	CE	RS	VT	N	CE	RS	VT	N	CE	RS	VT	N	CE
x_1	1	0.01	2	0	1	0.05	4	0	1	0.17	12	0	0	0.62	40	8
x_2	1	0.01	1	0	0	0.02	1	1	1	0.02	1	0	1	0.02	1	0
x_3	1	0.00	1	0	1	0.00	1	0	1	0.01	1	0	1	0.01	1	0
x_4	0	0.03	2	2	0	0.24	14	8	0	2.59	101	57	0	32.74	784	514
x_5	1	0.02	1	0	1	0.02	1	0	1	0.02	1	0	1	0.02	1	0
x_6	0	0.01	1	1	0	0.05	4	1	0	0.17	12	4	0	0.63	32	24
x_7	1	0.01	2	0	1	0.05	4	0	1	0.19	12	0	0	0.91	48	8
x_8	0	0.06	8	4	1	0.22	14	0	1	0.23	14	0	1	0.29	14	0
x_9	1	0.00	1	0	1	0.00	1	0	1	0.01	1	0	1	0.01	1	0
x_{10}	1	0.03	4	0	0	0.39	32	32	0	1.27	128	16	0	22.43	1024	1024
x_{11}	1	0.02	1	0	1	0.01	1	0	1	0.01	1	0	1	0.02	1	0
x_{12}	1	0.01	2	0	1	0.02	2	0	1	0.02	2	0	1	0.03	2	0
x_{13}	1	0.00	1	0	1	0.01	1	0	1	0.01	1	0	1	0.02	1	0
x_{14}	1	0.01	1	0	0	0.02	1	1	1	0.02	1	0	0	0.03	1	1
x_{15}	1	0.01	1	0	0	0.02	1	1	1	0.02	1	0	0	0.03	1	1
x_{16}	1	0.00	1	0	1	0.01	2	0	1	0.06	6	0	1	0.20	20	0
x_{17}	1	0.01	2	0	1	0.05	6	0	1	0.19	16	0	1	0.47	36	0
x_{18}	1	0.00	1	0	1	0.01	1	0	1	0.01	1	0	1	0.01	1	0
x_{19}	1	0.00	1	0	1	0.00	1	0	1	0.01	1	0	1	0.01	1	0
x_{20}	1	0.01	2	0	1	0.06	4	0	1	0.24	16	0	0	1.18	56	20
x_{21}	1	0.02	2	0	0	0.39	26	20	0	6.73	288	220	0	161.90	3194	2440
x_{22}	1	0.04	4	0	0	0.33	32	32	0	1.43	128	1	0	20.21	1024	1024
x_{23}	1	0.01	1	0	1	0.02	4	0	1	0.06	8	0	0	0.18	12	12
x_{24}	0	0.05	3	3	0	1.21	34	34	0	19.17	377	377	0	377.53	4181	4181
x_{25}	1	0.01	2	0	1	0.01	2	0	1	0.02	2	0	1	0.02	2	0

Table 1: Exact verification results for the small RNN in which *RS* is the verification result (*RS* = 1 → *ROBUST*, *RS* = 0 → *NOT_ROBUST*), *VT* is the verification time in seconds, *N* is the number of star sets in the output reachable set, and *CE* is the number of counterexample sets. Verification performance varies significantly for different input points.

5.2 Approximate Verification

The verification results using the approximate scheme, and RnnVerify are presented in Table 2. Full results can be found in the Appendix.

Conservativeness Improvement. We compare the number of provable cases over 25 input points between RnnVerify and our approximate verification scheme. For the small networks, i.e., $\mathcal{N}_{2,0}$ and $\mathcal{N}_{2,2}$, RnnVerify can prove that for all T_{max}, $2 \leq T_{max} \leq 20$, the network is robust for all input points. However, this is not the case in our results. There are some cases in which our method returns UNKNOWN results. For example, with the approximate reachability, we can prove that the network $N_{2,0}$ is robust only for 23 cases (over 25) at $T_{max} = 20$.

For larger networks ($\mathcal{N}_{4,0}, \mathcal{N}_{4,2}, \mathcal{N}_{4,4}$ and $\mathcal{N}_{8,0}$), our approximate verification outperforms RnnVerify in the number of provable cases when dealing with large T_{max}. For example, for $N_{4,0}$, at $T_{max} = 20$, the approximate verification can prove the robustness of the network for 24 input points even using full relaxation, i.e., RF = 1. (see Table 2) while RnnVerify proves only 12 cases. Even there are some concerns about RnnVerify results, we still want to know how much improvement in terms of conservativeness can be done by our approach compared with RnnVerify. To do that, we compute the total number of cases that can be proved by RnnVerify and our approach for 6 networks with 25 points for each (total number of queries is $6 \times 25 \times 19 = 2850$). From the full verification results (presented in the Appendix), RnnVerify can prove a total of 2191 cases while our approximate verification can prove 2463 cases without relaxation ($\approx$ 12.5% improvement) and 1978 cases with full relaxation ($\approx -9.7\%$ improvement). If we consider only 4 largest networks, RNNVerify can prove a total of 1241 cases while our approach can prove 1672 cases without relaxation ($\approx$ 35% improvement) and 1277 cases with full relaxation ($\approx$ 3% improvement).

Timing Improvement. From the full verification results, one can see that our approximate verification approach is much faster than RnnVerify approach in most of the cases except for $N_{2,2}$ and $N_{4,4}$ for large T_{max}, i.e., $T_{max} \geq 10$. Impressively, our approach can achieve up to 2407.6× faster without relaxation (network $N_{4,0}$ for $T_{max} = 19$). Our verification with full relaxation is always very fast for all networks. It can even be more than 5000× faster than RnnVerify as in case of $N_{4,0}$ with $T_{max} = 15$.

Relaxation Factor and disturbance bound effects. The timing performance of approximate verification without relaxation depends significantly on the size of the attack, i.g., the disturbance bound. As seen in Figure 5, increasing the disturbance bound ϵ or decreasing the relaxation factor RF increase the number of cases to be verified but reduces the verification time. With full relaxation, the time remains unchanged. This occurs as no optimization problems are solved for constructing the reachable set and verifying the robustness of the network.

6 RELATED WORK

Deep Neural Network Verification. Numerous techniques and tools have been proposed recently [10, 11, 17, 31]. These techniques can be categorized into sound and complete methods or only sound methods. Typical representatives of sound and complete methods

$\mathcal{N}$	T_{max}	RnnVerify		relax-star-RF=0				relax-star-RF=0.5				relax-star-RF=1			
		N_p	VT	N_p	r_c	VT	r_t	N_p	r_c	VT	r_t	N_p	r_c	VT	r_t
$\mathcal{N}_{2,0}$	5	25	3.13	25	0	0.02	173.3×	25	0	0.01	241.0×	23	−2	0.01	300.3×
	10	25	4.08	25	0	0.03	142.4×	25	0	0.03	150.9×	24	−1	0.02	195.9×
	15	25	4.61	24	−1	0.05	92.3×	24	−1	0.05	99.2×	23	−2	0.04	130.7×
	20	25	4.98	23	−2	0.08	65.7×	23	−2	0.07	71.5×	23	−2	0.05	94.5×
$\mathcal{N}_{2,2}$	5	25	16.72	23	−2	0.04	469.2×	23	−2	0.03	545.8×	21	−4	0.01	1216.3×
	10	25	5.74	21	−4	0.52	11.1×	21	−4	0.45	12.8×	13	−12	0.04	142.1×
	15	25	6.40	10	−15	12.69	0.5×	10	−15	15.31	0.4×	8	−17	0.11	59.3×
	20	25	8.31	9	−16	91.68	0.1×	8	−17	109.67	0.1×	6	−19	0.28	29.9×
$\mathcal{N}_{4,0}$	5	25	13.39	24	−1	0.01	898.4×	24	−1	0.01	927.7×	23	−2	0.01	1304.9×
	10	14	45.53	24	10	0.04	1069.2×	24	10	0.04	1128.6×	22	8	0.02	2097.1×
	15	12	176.54	24	12	0.08	2258.8×	24	12	0.07	2450.6×	24	12	0.04	4884.5×
	20	12	150.75	25	13	0.13	1119.9×	25	13	0.12	1216.9×	24	12	0.05	2834.8×
$\mathcal{N}_{4,2}$	5	19	32.93	22	3	0.08	414.2×	22	3	0.07	470.6×	16	−3	0.02	1837.0×
	10	16	144.69	20	4	0.46	317.8×	19	3	0.41	353.6×	15	−1	0.04	3498.9×
	15	16	11.01	18	2	2.78	4.0×	18	2	2.30	4.8×	15	−1	0.11	100.0×
	20	15	14.07	18	3	14.28	1.0×	18	3	10.41	1.4×	15	0	0.33	42.3×
$\mathcal{N}_{4,4}$	5	25	6.05	25	0	0.07	89.1×	25	0	0.06	100.9×	22	−3	0.02	271.7×
	10	8	1.16	25	17	4.17	0.3×	23	15	6.58	0.2×	14	6	0.05	24.1×
	15	8	1.81	25	17	69.82	0.0×	17	9	113.62	0.0×	13	5	0.14	13.3×
	20	8	2.61	21	13	509.02	0.0×	15	7	436.16	0.0×	12	4	0.43	6.0×
$\mathcal{N}_{8,0}$	5	2	2.54	22	20	0.10	24.4×	22	20	0.06	44.3×	13	11	0.01	214.8×
	10	2	6.44	17	15	0.81	8.0×	17	15	1.08	6.0×	8	6	0.02	260.6×
	15	2	10.82	20	18	5.03	2.2×	20	18	5.21	2.1×	12	10	0.04	255.1×
	20	2	14.37	17	15	10.60	1.4×	17	15	12.63	1.1×	9	7	0.07	220.7×

Table 2: Verification results for the RNN using the approximate and relaxed reachability. N_p is the number of provably robust cases (over 25 input points), VT is the verification time in seconds, r_c is the improvement in the provable number of cases (conservativeness improvement), and r_t is the improvement in the average verification time (time improvement).

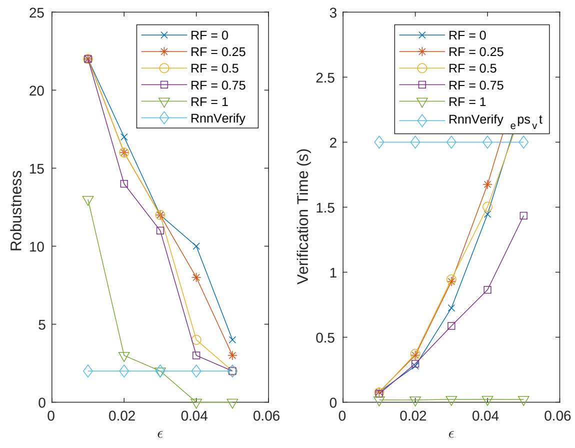

Figure 5: Verification performance with different attack bounds ϵ on network $\mathcal{N}_{8,0}$. Our approach using relaxed reachability significantly outperforms the RnnVerify in both number of provable cases and verification time. Our verification time increases along with ϵ except for using full relaxation, i.e., $RF = 1$. The number of provable cases reduces when we increase the relaxation factor (RF).

include the Satisfiability Modulo Theory (SMT) [13, 14], Mixed-Integer Linear Program (MILP) [19], reachability analysis [3, 28, 34]. As verifying DNNs is an NP-hard problem [13], sound and complete methods have limited scalability to deal with very large networks. To improve the scalability of DNNs Verification, overapproximation methods have been extensively explored. Various techniques have been developed for verifying nonlinear systems including optimization-based approaches [7], semidefinite programming [8], abstract interpretation [22, 25], relaxed convex programs [15], and overapproximate and relaxed star reachability [26, 28, 30].

RNN Verification. Our research in this paper is mainly inspired by the works done in RnnVerify [12] and RNSVerify [1]. In RNSVerify, the authors propose an unrolling method to *unroll* an RNN to an equivalent large FFNN whose verification problem can be encoded as a big MILP, which can be solved by the authors' previous proposed approach [19]. The unrolling technique is sound and complete but has a scalability issue because verifying a bounded n-steps RNN is equivalent to verifying $(n + 1)$-layers FFNN, where n is the length of a sequence of inputs. When the length of the sequence inputs increases, the unrolled FFNN network becomes very large and cannot be verified. Our exact verification (using exact reachability) is also sound and complete. However, we do not need to unroll the network when computing the exact reachable set. Instead, the influence of the previous hidden states is added directly into the current hidden state reachable sets by the Minkowski sum of star sets. In RnnVerify, [12], the authors verify RNNs using invariant inference without unrolling. Specifically, an FFNN with the same size as an RNN is created to over-approximate the RNN. Then, the RNN's properties are verified over this over-approximation using the SMT-based technique for FFNN verification [13, 14]. The authors leverage the well-studied notion of "inductive invariant" for constructing FFNN that encodes time-invariant properties of the RNN (i.e., input, hidden states, and outputs) in such a way that does not increase the size of the network as the unrolling method. Although the size of the FFNN is independent of the length of a sequence of inputs, its associating invariant needs to be refined when a new input comes to verify a property if the over-approximation is too coarse. Our approximate verification method constructs the invariant set of an RNN over finite time steps using approximate star reachability. However, unlike RnnVerify, which generates invariants that bound the values of all states in an RNN as *functions of time*, our invariant is a set of constraints representing the dependency between current states (i.e., hidden states or outputs) and previous states and inputs. New constraints between a new state (in a new time step) and the previous states (in the past) are added up to construct a new invariant via the Minkowski sum operation.

7 CONCLUSION

In this work, we present a complementary method for verifying recurrent neural networks based on extending the recent star set reachability. The algorithms have been implemented and made readily available through the NNV tool for the verification of neural networks. We demonstrate through experimental evaluation that the proposed approach is significantly faster and more capable than the state-of-the-art methods in RNN verification. As future work, we will extend our approach to LSTM and GRU recurrent networks inspired by the work done in [23]. We will also consider the verification problem of RNNs with respect to safety properties with temporal characteristics.

Acknowledgments. This work is partially supported by the National Science Foundation (NSF) under grant NSF-CCF-2220418 and the NSF Nebraska EPSCoR under grant OIA-2044049.

REFERENCES

[1] Michael E Akintunde, Andreea Kevorchian, Alessio Lomuscio, and Edoardo Pirovano. 2019. Verification of RNN-based neural agent-environment systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 6006–6013.

[2] Stanley Bak and Parasara Sridhar Duggirala. 2017. Simulation-equivalent reachability of large linear systems with inputs. In *International Conference on Computer Aided Verification*. Springer, 401–420.

[3] Stanley Bak, Hoang-Dung Tran, Kerianne Hobbs, and Taylor T. Johnson. 2020. Improved Geometric Path Enumeration for Verifying ReLU Neural Networks. In *32nd International Conference on Computer Aided Verification*. Springer.

[4] Xie Chen, Xunying Liu, Yongqiang Wang, Mark JF Gales, and Philip C Woodland. 2016. Efficient training and evaluation of recurrent neural network language models for automatic speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 24, 11 (2016), 2146–2157.

[5] Dong Dong, Xiao-Yang Li, and Fu-Qiang Sun. 2017. Life prediction of jet engines based on LSTM-recurrent neural networks. In *2017 Prognostics and system health management conference (PHM-Harbin)*. IEEE, 1–6.

[6] Parasara Sridhar Duggirala and Mahesh Viswanathan. 2016. Parsimonious, simulation based verification of linear systems. In *International Conference on Computer Aided Verification*. Springer, 477–494.

[7] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. 2018. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*. Springer, 121–138.

[8] Mahyar Fazlyab, Manfred Morari, and George J Pappas. 2020. Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. *IEEE Trans. Automat. Control* (2020).

[9] Yoav Goldberg. 2017. Neural network methods for natural language processing. *Synthesis lectures on human language technologies* 10, 1 (2017), 1–309.

[10] Navid Hashemi, Bardh Hoxha, Tomoya Yamaguchi, Danil Prokhorov, Geogios Fainekos, and Jyotirmoy Deshmukh. 2023. A Neurosymbolic Approach to the Verification of Temporal Logic Properties of Learning enabled Control Systems. *arXiv preprint arXiv:2303.05394* (2023).

[11] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. 2020. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review* 37 (2020), 100270.

[12] Yuval Jacoby, Clark Barrett, and Guy Katz. 2020. Verifying recurrent neural networks using invariant inference. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 57–74.

[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 *International Conference on Computer Aided Verification*. Springer, 97–117.

[14] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. 2019. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*. Springer.

[15] Haitham Khedr, James Ferlez, and Yasser Shoukry. 2021. Peregrinn: Penalized-relaxation greedy neural network verifier. In *International Conference on Computer Aided Verification*. Springer, 287–300.

[16] Ching-Yun Ko, Zhaoyang Lyu, Lily Weng, Luca Daniel, Ngai Wong, and Dahua Lin. 2019. POPQORN: Quantifying robustness of recurrent neural networks. In *International Conference on Machine Learning*. PMLR, 3468–3477.

[17] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, and Mykel J Kochenderfer. 2021. Algorithms for Verifying Deep Neural Networks. *Foundations and Trends® in Optimization* 4, 3-4 (2021), 244–404.

[18] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2016. Recurrent neural network for text classification with multi-task learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. 2873–2879.

[19] Alessio Lomuscio and Lalit Maganti. 2017. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351* (2017).

[20] Larry R Medsker and LC Jain. 2001. Recurrent neural networks. *Design and Applications* 5 (2001), 64–67.

[21] Oliver Obst. 2014. Distributed fault detection in sensor networks using a recurrent neural network. *Neural processing letters* 40, 3 (2014), 261–273.

[22] Pavithra Prabhakar and Zahra Rahimi Afzal. 2019. Abstraction based output range analysis for neural networks. *Advances in Neural Information Processing Systems* 32 (2019).

[23] Wonryong Ryou, Jiayu Chen, Mislav Balunovic, Gagandeep Singh, Andrei Marian Dan, and Martin T Vechev. 2020. Fast and effective robustness certification for recurrent neural networks. *CoRR abs/2005.13300* (2020).

[24] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. 2018. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*. 10825–10836.

[25] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 41.

[26] Hoang-Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T. Johnson. 2020. Verification of Deep Convolutional Neural Networks Using ImageStars. In *32nd International Conference on Computer-Aided Verification (CAV)*. Springer.

[27] Hoang-Dung Tran, Patrick Musau, Diego Manzanas Lopez, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T Johnson. 2019. Parallelizable reachability analysis algorithms for feed-forward neural networks. In *2019 IEEE/ACM 7th International Conference on Formal Methods in Software Engineering (FormaliSE)*. IEEE, 51–60.

[28] Hoang-Dung Tran, Patrick Musau, Diego Manzanas Lopez, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T. Johnson. 2019. Star-Based Reachability Analsysis for Deep Neural Networks. In *23rd International Symposisum on Formal Methods (FM'19)*. Springer International Publishing.

[29] Hoang-Dung Tran, Neelanjana Pal, Diego Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T Johnson. 2021. Verification of piecewise deep neural networks: a star set approach with zonotope pre-filter. *Formal Aspects of Computing* 33, 4 (2021), 519–545.

[30] Hoang-Dung Tran, Neelanjana Pal, Patrick Musau, Diego Manzanas Lopez, Nathaniel Hamilton, Xiaodong Yang, Stanley Bak, and Taylor T Johnson. 2021. Robustness verification of semantic segmentation neural networks using relaxed reachability. In *International Conference on Computer Aided Verification*. Springer.

[31] Hoang-Dung Tran, Weiming Xiang, and Taylor T Johnson. 2020. Verification approaches for learning-enabled autonomous cyber-physical systems. *IEEE Design & Test* (2020).

[32] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T Johnson. 2020. NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *Computer Aided Verification: 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020*. Springer, 3–17.

[33] Weiming Xiang, Patrick Musau, Ayana A Wild, Diego Manzanas Lopez, Nathaniel Hamilton, Xiaodong Yang, Joel Rosenfeld, and Taylor T Johnson. 2018. Verification for machine learning, autonomy, and neural networks survey. *arXiv preprint arXiv:1810.01989* (2018).

[34] Xiaodong Yang, Taylor T Johnson, Hoang-Dung Tran, Tomoya Yamaguchi, Bardh Hoxha, and Danil V Prokhorov. 2021. Reachability analysis of deep ReLU neural networks using facet-vertex incidence.. In *HSCC*. 18–1.

[35] Hongce Zhang, Maxwell Shinn, Aarti Gupta, Arie Gurfinkel, Nham Le, and Nina Narodytska. 2020. Verification of recurrent neural networks for cognitive tasks via reachability analysis. In *ECAI 2020*. IOS Press, 1690–1697.

APPENDIX

$\mathcal{N}$	T_{max}	RnnVerify		relax-star-RF=0				relax-star-RF=0.5				relax-star-RF=1			
		N_p	VT	N_p	r_c	VT	r_t	N_p	r_c	VT	r_t	N_p	r_c	VT	r_t
$\mathcal{N}_{2,0}$	2	25	0.18	25	0	0.01	21.4×	25	0	0.01	30.4×	24	−1	0.00	37.5×
	3	25	2.23	25	0	0.01	280.8×	25	0	0.01	297.8×	25	0	0.01	350.6×
	4	25	2.82	25	0	0.01	289.9×	25	0	0.01	292.0×	24	−1	0.01	332.9×
	5	25	3.13	25	0	0.02	203.5×	25	0	0.01	231.5×	23	−2	0.01	290.6×
	6	25	3.39	25	0	0.02	217.3×	25	0	0.02	220.4×	25	0	0.01	259.7×
	7	25	3.61	24	−1	0.02	189.0×	24	−1	0.02	192.5×	24	−1	0.02	234.1×
	8	25	3.74	22	−3	0.02	162.1×	22	−3	0.02	163.9×	22	−3	0.02	207.0×
	9	25	3.93	24	−1	0.03	148.7×	24	−1	0.03	151.6×	23	−2	0.02	188.9×
	10	25	4.08	25	0	0.03	133.3×	25	0	0.03	136.1×	24	−1	0.02	174.8×
	11	25	4.21	24	−1	0.04	118.2×	24	−1	0.03	121.8×	24	−1	0.03	160.3×
	12	25	4.26	24	−1	0.04	108.9×	24	−1	0.04	114.6×	24	−1	0.03	145.3×
	13	25	4.41	23	−2	0.04	101.3×	23	−2	0.04	106.0×	22	−3	0.03	132.7×
	14	25	4.53	24	−1	0.05	93.9×	24	−1	0.05	97.7×	24	−1	0.04	120.7×
	15	25	4.61	24	−1	0.05	86.6×	24	−1	0.05	90.0×	23	−2	0.04	112.5×
	16	25	4.71	24	−1	0.06	79.1×	24	−1	0.06	83.8×	24	−1	0.04	104.7×
	17	25	4.79	23	−2	0.06	73.7×	23	−2	0.06	77.6×	22	−3	0.05	96.8×
	18	25	4.81	25	0	0.07	68.7×	25	0	0.07	71.6×	24	−1	0.05	91.8×
	19	25	4.86	25	0	0.08	64.6×	25	0	0.07	67.1×	24	−1	0.06	86.3×
	20	25	4.98	23	−2	0.08	60.9×	23	−2	0.08	63.9×	23	−2	0.06	82.8×
$\mathcal{N}_{4,0}$	2	25	0.28	25	0	0.01	28.8×	25	0	0.01	43.6×	25	0	0.01	54.9×
	3	25	9.34	25	0	0.01	1109.0×	25	0	0.01	1163.5×	25	0	0.01	1411.9×
	4	25	14.87	23	−2	0.01	1177.7×	23	−2	0.01	1230.0×	23	−2	0.01	1655.8×
	5	25	13.39	24	−1	0.02	814.8×	24	−1	0.02	857.6×	23	−2	0.01	1203.1×
	6	25	26.75	24	−1	0.03	1011.3×	24	−1	0.02	1299.8×	22	−3	0.01	2020.4×
	7	25	35.27	25	0	0.03	1293.5×	25	0	0.03	1346.8×	22	−3	0.02	2229.2×
	8	25	26.95	24	−1	0.04	758.5×	24	−1	0.03	791.9×	21	−4	0.02	1466.9×
	9	23	47.98	25	2	0.04	1168.3×	25	2	0.04	1245.5×	22	−1	0.02	2288.7×
	10	14	45.53	24	10	0.05	986.9×	24	10	0.04	1041.9×	22	8	0.02	1911.7×
	11	14	48.78	25	11	0.05	924.9×	25	11	0.05	987.7×	23	9	0.03	1836.8×
	12	14	33.55	25	11	0.06	545.2×	25	11	0.06	583.2×	23	9	0.03	1135.0×
	13	13	127.67	25	12	0.07	1830.6×	25	12	0.06	1985.5×	24	11	0.03	3862.7×
	14	12	105.00	25	13	0.08	1361.3×	25	13	0.07	1481.5×	24	12	0.04	2901.7×
	15	12	176.54	24	12	0.08	2096.7×	24	12	0.08	2274.3×	24	12	0.04	4464.5×
	16	12	181.04	24	12	0.09	1954.9×	24	12	0.08	2153.3×	24	12	0.04	4220.7×
	17	12	130.32	25	13	0.10	1321.3×	25	13	0.09	1454.8×	25	13	0.05	2794.8×
	18	12	204.55	25	13	0.10	1951.9×	25	13	0.10	2139.1×	25	13	0.05	4039.7×
	19	12	271.00	24	12	0.12	2177.5×	24	12	0.12	2311.0×	23	11	0.05	4981.4×
	20	12	150.75	25	13	0.14	1055.7×	25	13	0.13	1144.8×	24	12	0.06	2604.5×
$\mathcal{N}_{4,2}$	2	25	0.34	23	−2	0.02	19.5×	23	−2	0.01	27.7×	20	−5	0.01	39.7×
	3	25	23.51	21	−4	0.02	1465.2×	21	−4	0.01	1664.4×	21	−4	0.01	3210.6×
	4	23	49.02	20	−3	0.03	1853.9×	20	−3	0.02	2023.1×	17	−6	0.01	4875.3×
	5	19	32.93	22	3	0.07	461.5×	22	3	0.07	493.5×	16	−3	0.01	2405.2×
	6	17	46.90	21	4	0.10	454.7×	21	4	0.10	485.2×	16	−1	0.02	2644.0×
	7	17	39.89	22	5	0.15	272.6×	22	5	0.14	292.2×	20	3	0.02	1760.4×
	8	16	57.42	20	4	0.22	266.9×	20	4	0.20	284.3×	15	−1	0.03	2071.1×
	9	16	87.27	20	4	0.31	277.2×	20	4	0.28	306.6×	16	0	0.03	2564.1×
	10	16	144.69	20	4	0.45	323.9×	19	3	0.40	361.2×	15	−1	0.04	3493.5×
	11	16	87.08	18	2	0.64	136.7×	18	2	0.57	151.8×	16	0	0.05	1724.2×
	12	16	8.94	20	4	0.93	9.6×	20	4	0.82	10.9×	14	−2	0.06	145.8×
	13	16	9.80	21	5	1.35	7.3×	21	5	1.15	8.5×	14	−2	0.07	131.2×
	14	16	10.07	20	4	1.93	5.2×	19	3	1.61	6.3×	15	−1	0.09	110.8×
	15	16	11.01	18	2	2.74	4.0×	18	2	2.28	4.8×	15	−1	0.11	99.9×

	16	16	11.36	18	2	3.89	2.9×	18	2	3.17	3.6×	15	−1	0.13	84.6×
	17	15	12.09	18	3	5.47	2.2×	18	3	4.40	2.7×	15	0	0.16	74.1×
	18	15	13.05	18	3	7.52	1.7×	18	3	5.92	2.2×	15	0	0.20	64.9×
	19	15	13.23	16	1	10.38	1.3×	16	1	7.86	1.7×	14	−1	0.26	51.2×
	20	15	14.07	18	3	14.23	1.0×	18	3	10.34	1.4×	15	0	0.33	42.8×
	2	25	0.47	25	0	0.01	46.0×	25	0	0.01	65.1×	25	0	0.01	32.2×
$\mathcal{N}_{4,4}$	3	25	11.97	25	0	0.02	741.4×	25	0	0.02	783.7×	24	−1	0.01	1546.1×
	4	25	23.44	25	0	0.03	682.4×	25	0	0.03	738.5×	22	−3	0.01	2264.3×
	5	25	6.05	25	0	0.07	91.8×	25	0	0.06	103.7×	22	−3	0.01	425.6×
	6	25	2.13	25	0	0.22	9.6×	25	0	0.24	8.8×	19	−6	0.02	112.8×
	7	8	0.87	24	16	0.43	2.0×	24	16	0.48	1.8×	14	6	0.02	36.3×
	8	8	0.93	25	17	0.86	1.1×	24	16	1.45	0.6×	16	8	0.03	31.4×
	9	8	1.02	25	17	2.31	0.4×	24	16	3.56	0.3×	14	6	0.04	27.6×
	10	8	1.16	25	17	4.15	0.3×	23	15	6.59	0.2×	14	6	0.05	24.8×
	11	8	1.28	25	17	7.95	0.2×	21	13	14.52	0.1×	13	5	0.06	21.9×
	12	8	1.39	25	17	14.58	0.1×	20	12	29.56	0.0×	13	5	0.07	19.3×
	13	8	1.47	24	16	25.38	0.1×	18	10	48.97	0.0×	12	4	0.09	16.4×
	14	8	1.61	25	17	43.80	0.0×	19	11	74.46	0.0×	14	6	0.11	14.7×
	15	8	1.81	25	17	69.06	0.0×	17	9	114.09	0.0×	13	5	0.14	13.4×
	16	8	2.03	24	16	111.60	0.0×	17	9	162.07	0.0×	12	4	0.18	11.3×
	17	8	2.08	23	15	168.53	0.0×	16	8	215.15	0.0×	13	5	0.21	9.7×
	18	8	2.32	23	15	249.99	0.0×	16	8	277.70	0.0×	12	4	0.28	8.4×
	19	8	2.62	23	15	360.54	0.0×	16	8	353.41	0.0×	14	6	0.34	7.6×
	20	8	2.61	21	13	509.02	0.0×	15	7	436.16	0.0×	12	4	0.43	6.0×
	2	25	0.61	24	−1	0.01	78.8×	24	−1	0.01	83.8×	23	−2	0.00	133.0×
$\mathcal{N}_{8,0}$	3	5	15.10	23	18	0.02	941.4×	23	18	0.02	982.3×	15	10	0.01	2505.7×
	4	2	1.81	22	20	0.03	57.6×	22	20	0.03	61.5×	13	11	0.01	218.3×
	5	2	2.54	22	20	0.06	44.0×	22	20	0.06	45.9×	13	11	0.01	241.6×
	6	2	3.29	21	19	0.10	31.9×	21	19	0.12	26.3×	10	8	0.01	256.3×
	7	2	4.15	16	14	0.20	21.1×	16	14	0.24	17.6×	7	5	0.02	270.7×
	8	2	5.00	19	17	0.29	17.0×	19	17	0.40	12.5×	7	5	0.02	279.1×
	9	2	5.52	19	17	0.47	11.6×	17	15	0.61	9.0×	9	7	0.02	266.6×
	10	2	6.44	17	15	0.80	8.1×	17	15	1.07	6.0×	8	6	0.02	273.5×
	11	2	7.15	20	18	1.17	6.1×	20	18	1.46	4.9×	10	8	0.03	269.1×
	12	2	8.55	20	18	1.71	5.0×	20	18	2.11	4.0×	11	9	0.03	287.3×
	13	2	8.89	20	18	2.45	3.6×	20	18	2.87	3.1×	9	7	0.03	265.2×
	14	2	9.97	19	17	3.67	2.7×	18	16	3.74	2.7×	9	7	0.04	269.2×
	15	2	10.82	20	18	5.00	2.2×	20	18	5.17	2.1×	12	10	0.04	262.6×
	16	2	11.42	18	16	6.48	1.8×	18	16	6.49	1.8×	9	7	0.04	254.1×
	17	2	12.33	17	15	7.85	1.6×	17	15	8.14	1.5×	11	9	0.05	250.9×
	18	2	12.94	15	13	8.97	1.4×	15	13	9.86	1.3×	10	8	0.05	244.0×
	19	2	14.20	15	13	9.78	1.5×	15	13	11.02	1.3×	11	9	0.06	249.7×
	20	2	14.37	17	15	10.53	1.4×	17	15	12.54	1.1×	9	7	0.06	233.8×

Table 3: Full verification results. N_p is the number of provably robust cases (over 25 input points), VT is the verification time in seconds, r_c is the improvement in the provable number of cases (conservativeness improvement), and r_t is the improvement in the average verification time (time improvement).