Liang Z, Liu WW, Song F *et al.* Qualitative and quantitative model checking against recurrent neural networks. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 39(6): 1292-1311 Nov. 2024. DOI: 10.1007/s11390-023-2703-2

# Qualitative and Quantitative Model Checking Against Recurrent Neural Networks

Zhen Liang[1] (梁　震), Wan-Wei Liu[2,*] (刘万伟), *Senior Member, CCF*
Fu Song[3] (宋　富), *Senior Member, CCF*, Bai Xue[4] (薛　白), Wen-Jing Yang[1] (杨文婧)
Ji Wang[1] (王　戟), *Fellow, CCF*, and Zheng-Bin Pang[2] (庞征斌)

[1] *Institute for Quantum Information & State Key Laboratory of High Performance Computing, College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China*

[2] *College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China*

[3] *School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China*

[4] *Institute of Software, Chinese Academy of Sciences, Beijing 100190, China*

E-mail: liangzhen@nudt.edu.cn; wwliu@nudt.edu.cn; songfu@shanghaitech.edu.cn; xuebai@ios.ac.cn
wenjing.yang@nudt.edu.cn; wj@nudt.edu.cn; zhengbinpang@nudt.edu.cn

Received July 23, 2022; accepted July 10, 2023.

**Abstract** Recurrent neural networks (RNNs) have been heavily used in applications relying on sequence data such as time series and natural languages. As a matter of fact, their behaviors lack rigorous quality assurance due to the black-box nature of deep learning. It is an urgent and challenging task to formally reason about the behaviors of RNNs. To this end, we first present an extension of linear-time temporal logic to reason about properties with respect to RNNs, such as local robustness, reachability, and some temporal properties. Based on the proposed logic, we formalize the verification obligation as a Hoare-like triple, from both qualitative and quantitative perspectives. The former concerns whether all the outputs resulting from the inputs fulfilling the pre-condition satisfy the post-condition, whereas the latter is to compute the probability that the post-condition is satisfied on the premise that the inputs fulfill the pre-condition. To tackle these problems, we develop a systematic verification framework, mainly based on polyhedron propagation, dimension-preserving abstraction, and the Monte Carlo sampling. We also implement our algorithm with a prototype tool and conduct experiments to demonstrate its feasibility and efficiency.

**Keywords** recurrent neural network, model checking, temporal logic, qualitative/quantitative verification

## 1 Introduction

Neural networks (NNs) have achieved remarkable performance in a variety of challenging tasks in the past few years, such as image recognition[1], natural language processing[2], speech processing[3], and multi-view clustering[4]. Therefore, it is desired to deploy neural networks in safety-critical applications as well, for instance, autonomous driving[5] and medical diagnostics[6]. However, various concerns have arisen due to the black-box nature of neural networks, which hinders their applications in safety-critical domains[7]. For example, it is extremely difficult to explain the decisions of a neural network[8]. Furthermore, neural networks have been shown to be fundamentally vulnerable to minor input perturbations. It means that a small perturbation to a correctly handled input may induce unexpected results[9, 10]. Therefore, there is a pressing need to formally reason about the behaviors of neural networks before deploying them.

Recently, dozens of verification approaches for neural networks have been proposed[11–14] by leverag-

Regular Paper

This work was supported by the National Natural Science Foundation of China under Grant Nos. 61872371, 62032024, and U19A2062, and the Open Fund from the State Key Laboratory of High Performance Computing of China (HPCL) under Grant No. 202001-07.

*Corresponding Author

ing techniques such as constraint solving and abstract interpretation. However, all these approaches are designated for verifying feed-forward neural networks such as fully connected feed-forward networks (FNNs) and convolutional neural networks (CNNs), and cannot be directly applied to verifying recurrent neural networks (RNNs). Indeed, in contrast to feed-forward neural networks which only consider the current input, an RNN considers the current input and intermediate results of the previous computation, namely that it can memorize previous computing results based on its internal memory over sequential inputs. Until recently, some efforts have been made to verify RNNs[15–21]. Unfortunately, almost all the existing methods are limited in qualitatively verifying properties in linear constraints (e.g., robustness), and largely ignoring the sequential nature of inputs/outputs of RNNs. To the best of our knowledge, only [15] and [21] support qualitative reasoning about RNNs using temporal logic over input/output sequences and only [22] supports quantitative RNN verification against RNNs. Though temporal properties could be expressed in linear constraints when the lengths of input/output sequences are bounded, it is more intuitive and succinct to express properties in temporal logic and the verification approaches provide a generic framework for reasoning about the behaviors of neural networks[14].

In order to reason about the behaviors of RNNs, we in this paper present a temporal logic, called $\mathrm{LTL}_f[\boldsymbol{x}]$, which is an extension of $\mathrm{LTL}_f$ (i.e., linear-time temporal logic on finite traces), where $[\boldsymbol{x}]$ addresses the special symbol $\boldsymbol{x}$ corresponding to the designated vector in a vector sequence. This logic is able to specify properties such as robustness, reachability, and other temporal properties over input/output sequences. We formalize the verification problem of an RNN $\mathcal{N}$ as a Hoare-like triple, $(\varphi\{\mathcal{N}\}\psi)_k$, where the pre-condition $\varphi$ (resp. post-condition $\psi$) describes the property of the input (resp. output) sequences, and $k$ is the length bound of the input sequences. We consider both the qualitative and quantitative perspectives of the verification problem, where the former concerns whether all the outputs resulting from the inputs fulfilling the pre-condition satisfy the post-condition, and the latter is to compute the probability (or, ratio) of the satisfaction of the post-condition conditioned by the pre-condition. Arguably, quantitative verification is far more useful[23], since it could provide a probabilistic guarantee of the behaviors of RNNs.

Why do we concern about recurrent networks? First and foremost, such neural networks have important applications in many fields, such as computer vision[24], natural language processing[25], and speech processing[3]. Second, an FNN is a special case of an RNN; namely, an FNN can be regarded as an RNN without "looping". Last but not least, as an RNN deals with data sequences, there exist important temporal patterns in the input/output ends.

Model checking of NNs is never straightforward, and the difficulty partially lies in the large quantity of parameters and the complexity of the data processing procedure during computation. Moreover, in addition to an affine transformation, each layer also includes a non-linear activation function, which enables an NN to fit a (continuous) function with arbitrary precision[26].

From the algebra perspective, using a rectified linear unit (ReLU) as the activation function for NNs may make them easier to deal with. In this case, an NN essentially establishes a piece-wise linear mapping. Thus, if the input space is a union of several polyhedra (called a bundle), the output space is also a bundle. Indeed, given an $\mathrm{LTL}_f[\boldsymbol{x}]$ formula $\varphi$, the collection of all vector sequences initially satisfying $\varphi$ is isomorphic to a bundle.

When performing model checking on RNN against $\mathrm{LTL}_f[\boldsymbol{x}]$, there are several technical challenges.

1) First of all, a polyhedron can be uniquely determined via a set of vertices and extreme directions (a polyhedron is said to be unbounded whenever the direction set is not empty), and we need to compute a series of intermediate polyhedra, which is called forward propagation. It can be found that the number of vertices increases dramatically during the forward propagation. To alleviate this issue, we need to find a "larger" (or, an abstract) polyhedron with fewer vertices to tightly enclose it. The challenging part is that the abstracted one must have the same dimension, the reason of which will be explained in Section 4.

2) For a polyhedron, we need to perform abstraction to relax the number-explosion of vertices, which might lead to "infeasible counterexamples". Thus, refinement is required. Meanwhile, we need to perform the quantitative computation. Here, we use the Monte Carlo approach, namely, doing sampling when computing the ratio, which induces the problem that if we abstract the polyhedron by replacing it with one of a higher dimension, it will result in a 0/0 dilemma, be-

cause the probability of obtaining a sample within the feasible area is 0.

3) To determine the feasibility of a "point" (corresponding to a sequence) within the output space, we need to do the "backward propagation", for which we have to find a way to compute the preimage of the transformation.

4) Given that a polyhedron may be unbounded, performing abstraction must be more cautious. We must ensure that the volume of the increased part must be an infinitesimal value in comparison to the concrete one.

5) Lastly, it should be pointed out that we in general simultaneously have more than one polyhedron during the verification process. These polyhedra may have different dimensions. Though the polyhedra with the largest dimension may dominate the final result, we cannot discard those with lower dimensions in the intermediate steps, as a polyhedron's dimension may be lowered after propagation whenever the transformation is degenerated.

In this paper, we show how to address the above challenges and present a systematic model checking framework. In addition, to the best of our knowledge, this is the first work unifying qualitative and quantitative verification of RNNs and we implement a prototype, called Bidirectional Propagation & Monte Carlo Based Model Checker (BPMC$^2$), and the presented framework is experimentally evaluated with respect to BPMC$^2$.

The remainder of this paper is organized as follows. Section 2 introduces some basic notions and notations related to recurrent neural networks. In Section 3, we formally define the syntax and semantics of LTL$_f[\boldsymbol{x}]$, and the goals of qualitative and quantitative model checking. We elaborate the technical details for verification in Section 4, and experimental results are provided in Section 5. In Section 6, we discuss some related work on RNN verification. Finally, we conclude the paper in Section 7.

## 2 Preliminaries

### 2.1 Vectors and Operations

In this paper, we use lowercase letters like $a$, $b$, $c$, $c_1$, and $c_2$ to range over scalars, use bold lowercase letters such as $\boldsymbol{b}$, $\boldsymbol{b}_1$, and $\boldsymbol{x}$ to range over vectors, and use bold uppercase letters to refer to matrices, such as $\boldsymbol{M}$, $\boldsymbol{M}_1$, and $\boldsymbol{N}$. For each vector $\boldsymbol{b}$ (resp. matrix $\boldsymbol{M}$), we denote $\boldsymbol{b}[i]$ (resp. $\boldsymbol{M}[i]$) as the $(i+1)$-th scalar of $\boldsymbol{b}$ (resp. the $(i+1)$-th row of $\boldsymbol{M}$). We denote the set of positive real numbers as $\mathbb{R}_{>0}$. As usual, we denote the kernel of $\boldsymbol{M}$ as $\mathrm{Ker}\boldsymbol{M}$, which is the set $\{\boldsymbol{v} \mid \boldsymbol{M}\boldsymbol{v} = 0\}$.

Given a set of vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$, we call $\boldsymbol{b}$ a non-negative combination of them if $\boldsymbol{b} = \sum_{i=1}^{n} c_i \boldsymbol{b}_i$, where each $c_i \geqslant 0$. If in addition $\sum_{i=1}^{n} c_i = 1$ holds, we call $\boldsymbol{b}$ a convex-combination of $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$. Meanwhile, we denote $\mathrm{span}\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}$ for the set $\{\sum_{i=1}^{n} a_i \boldsymbol{b}_i\}$, namely, the linear space spanned by $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$.

Vector sequences are the heavily used mathematical structures in this paper and we use Greek bold letters to range over them. For a vector sequence $\boldsymbol{\tau} = \boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_k$, we write $\boldsymbol{\tau} \in \mathbb{R}^{n_1, n_2, \ldots, n_k}$ if $\boldsymbol{b}_i \in \mathbb{R}^{n_i}$, and let $\mathrm{len}(\boldsymbol{\tau}) = k$. Particularly, we write $\boldsymbol{\tau} \in (\mathbb{R}^n)^k$ if each $n_i = n$. For convenience, we sometimes directly view $\boldsymbol{\tau}$ as a common vector, which is the juxtaposition of all $\boldsymbol{b}_i$s. Namely, $\boldsymbol{\tau}$ is isomorphic to $(\boldsymbol{b}_1^{\mathrm{T}}, \boldsymbol{b}_2^{\mathrm{T}}, \ldots, \boldsymbol{b}_k^{\mathrm{T}})^{\mathrm{T}}$, denoted as $\mathrm{Jux}(\boldsymbol{\tau})$.

Given a matrix $\boldsymbol{M} \in \mathbb{R}^{m \times n}$, let $\mathrm{Ran}\boldsymbol{M} = \{\boldsymbol{M}\boldsymbol{b} \mid \boldsymbol{b} \in \mathbb{R}^n\}$ and let $\mathrm{Ker}\boldsymbol{M} = \{\boldsymbol{b} \in \mathbb{R}^n \mid \boldsymbol{M}\boldsymbol{b} = 0\}$, which are called the range and the kernel of $\boldsymbol{M}$, respectively.

From the standard theory of linear algebra, each real matrix (not necessarily square) $\boldsymbol{M}$ can be decomposed as $\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}$ where $\boldsymbol{U}$ and $\boldsymbol{V}$ are unitary matrices, and $\boldsymbol{D}$ is a quasi-diagonal matrix, i.e., $\boldsymbol{D} = \begin{pmatrix} \boldsymbol{D}' & 0 \\ 0 & 0 \end{pmatrix}$, where $\boldsymbol{D}' = \mathrm{diag}(\mathrm{b}_1, \ldots, \mathrm{b}_k)$ and each $b_i > 0$. Here, $\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}$ is called the singular value decomposition (SVD) of $\boldsymbol{M}$. In this case, we denote the matrix $\boldsymbol{V}^{\mathrm{T}} \begin{pmatrix} (\boldsymbol{D}')^{-1} & 0 \\ 0 & 0 \end{pmatrix} \boldsymbol{U}^{\mathrm{T}}$ by $\boldsymbol{M}^{+}$, called the Moore-Penrose inverse of $\boldsymbol{M}$.

Supposing $\boldsymbol{M} \in \mathbb{R}^{m \times n}$ and its SVD is given as above, we have $\boldsymbol{U} \in \mathbb{R}^{m \times m}$ and $\boldsymbol{V} \in \mathbb{R}^{n \times n}$, and also suppose that $\boldsymbol{D}' \in \mathbb{R}^{k \times k}$, where $k \leqslant \min\{m, n\}$. Then, both $\mathrm{Ker}\boldsymbol{M}$ and $\mathrm{Ran}\boldsymbol{M}$ can be succinctly represented as follows. Let $\boldsymbol{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m)$ and $\boldsymbol{V}^{\mathrm{T}} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n)$, and then we have $\mathrm{Ran}\boldsymbol{M} = \mathrm{span}\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k\}$ and $\mathrm{Ker}\boldsymbol{M} = \mathrm{span}\{\boldsymbol{v}_{k+1}, \ldots, \boldsymbol{v}_n\}$.

An important application of the Moore-Penrose inverse is to represent the preimage space of linear mappings.

**Theorem 1.** *The solution space of $\boldsymbol{M}\boldsymbol{x} \sim \boldsymbol{b}$ is not empty*[①] *if and only if*

[①]Here, $\sim$ can be any comparator like $>$, $\geqslant$, $<$, $\leqslant$, $\neq$ and so on.

$$\mathcal{V} = \mathrm{Ker}(\boldsymbol{I} - \boldsymbol{M}\boldsymbol{M}^{+}) \cap \{\boldsymbol{c} \mid \boldsymbol{c} \sim \boldsymbol{b}\} \neq \emptyset,$$

*and in this case, the solution space is* $\{\boldsymbol{M}^{+}\boldsymbol{c} + \boldsymbol{d} \mid \boldsymbol{c} \in \mathcal{V}, \boldsymbol{d} \in \mathrm{Ker}\boldsymbol{M}\}$.

*Proof.* According to the construction of $\boldsymbol{M}^{+}$, one can immediately examine that

$$\boldsymbol{M}\boldsymbol{M}^{+}\boldsymbol{M} = \boldsymbol{M} \quad \text{and} \quad (\boldsymbol{M}\boldsymbol{M}^{+})^{\mathrm{T}} = \boldsymbol{M}\boldsymbol{M}^{+}$$

hold. Suppose that $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ and let $\mathcal{U} = \{\boldsymbol{c} \in \mathbb{R}^{m} \mid \boldsymbol{c} \sim \boldsymbol{b}\}$, then for any $\boldsymbol{d}, \boldsymbol{d}' \in \mathbb{R}^{n}$ we have:

$$\begin{aligned}((\boldsymbol{I} - \boldsymbol{M}\boldsymbol{M}^{+})\boldsymbol{d})^{\mathrm{T}}(\boldsymbol{M}\boldsymbol{d}') &= \boldsymbol{d}^{\mathrm{T}}(\boldsymbol{I} - \boldsymbol{M}\boldsymbol{M}^{+})\boldsymbol{M}\boldsymbol{d}' \\ &= \boldsymbol{d}^{\mathrm{T}}(\boldsymbol{M} - \boldsymbol{M})\boldsymbol{d}' = 0,\end{aligned}$$

and this indicates that $(\boldsymbol{I} - \boldsymbol{M}\boldsymbol{M}^{+})\boldsymbol{d} \in (\mathrm{Ran}\boldsymbol{M})^{\perp}$ for any $\boldsymbol{d} \in \mathbb{R}^{n}$, because $\boldsymbol{M}\boldsymbol{d}'$ ranges over $\mathrm{Ran}\boldsymbol{M}$.

For each $\boldsymbol{x}$ having $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{c} \in \mathcal{U}$, we have that $\boldsymbol{c} \in \mathrm{Ran}\boldsymbol{M}$. Nevertheless, we on the other hand have

$$\boldsymbol{c} = \boldsymbol{M}\boldsymbol{M}^{+}\boldsymbol{c} + (\boldsymbol{I} - \boldsymbol{M}\boldsymbol{M}^{+})\boldsymbol{c},$$

hence we have $\boldsymbol{c} = \boldsymbol{M}\boldsymbol{M}^{+}\boldsymbol{c}$ and $(\boldsymbol{I} - \boldsymbol{M}\boldsymbol{M}^{+})\boldsymbol{c} = 0$, because $(\boldsymbol{I} - \boldsymbol{M}\boldsymbol{M}^{+})\boldsymbol{c} \in (\mathrm{Ran}\boldsymbol{M})^{\perp}$. In other words, we have $\boldsymbol{c} \in \mathrm{Ker}(\boldsymbol{I} - \boldsymbol{M}\boldsymbol{M}^{+}) \cap \mathcal{U} = \mathcal{V}$.

In this case, the solution space w.r.t. $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{c} = \boldsymbol{M}\boldsymbol{M}^{+}\boldsymbol{c}$ is definitely

$$\{\boldsymbol{M}^{+}\boldsymbol{c} + \boldsymbol{d} \mid \boldsymbol{d} \in \mathrm{Ker}\boldsymbol{M}\},$$

and this concludes the proof. □

**Corollary 1.** *The solution space of* $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{b}$ *is not empty if and only if*

$$(\boldsymbol{I} - \boldsymbol{M}\boldsymbol{M}^{+})\boldsymbol{b} = 0,$$

*and in this case, the solution space is* $\{\boldsymbol{M}^{+}\boldsymbol{b} + \boldsymbol{c} \mid \boldsymbol{c} \in \mathrm{Ker}\boldsymbol{M}\}$.

Each mapping $f\colon \mathbb{R}^{m} \to \mathbb{R}$ can be lifted into its vectorized form $f\colon (\mathbb{R}^{n})^{m} \to \mathbb{R}^{n}$. That is,

$$f(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_m) = (c_1, \ldots, c_n)^{\mathrm{T}},$$

where $\boldsymbol{b}_j = (b_{j,\,1}, \ldots,\ b_{j,\,n})^{\mathrm{T}}$ and $c_i = f(b_{1,\,i}, \ldots,\ b_{n,\,i})$. For convenience, for a mapping $f$ and a space $\mathcal{V}$, we let $f(\mathcal{V})$ be the space $\{f(\boldsymbol{v}) \mid \boldsymbol{v} \in \mathcal{V}\}$.

In this paper, we are particularly concerned with the $\boldsymbol{ReLU}$ function, defined as

$$\boldsymbol{ReLU}(x) = \begin{cases} x, & x \geqslant 0, \\ 0, & x < 0. \end{cases}$$

Meanwhile, let $n > 0$, for each $X \subseteq \{1, 2, \ldots, n\}$, and then we have the function $\boldsymbol{ReLU}_X : \mathbb{R}^{n} \to \mathbb{R}^{n}$, defined as $\boldsymbol{ReLU}_X((b_1, \ldots, b_n)^{\mathrm{T}}) = (c_1, \ldots, c_n)^{\mathrm{T}}$, where

$$c_i = \begin{cases} b_i, & i \notin X, \\ \dfrac{b_i + \mathrm{abs}(b_i)}{2}, & i \in X. \end{cases}$$

Hence, the vectorized lifting of $\boldsymbol{ReLU}$ coincides with $\boldsymbol{ReLU}_{\{1,\ \ldots,\ n\}}$ w.r.t. the domain $\mathbb{R}^{n}$. In what follows, we directly write $\boldsymbol{ReLU}_{\{i\}}$ as $\boldsymbol{ReLU}_i$.

Meanwhile, another operator utilized in the paper is the projection operator $\boldsymbol{Proj}_i$. For a vector $\boldsymbol{b} = (b_1, \ldots, b_n)^{\mathrm{T}}$, we define that $\boldsymbol{Proj}_i(\boldsymbol{d}) = (b_1, \ldots, b_{i-1}, 0, b_i, \ldots, b_n)^{\mathrm{T}}$. Then, $\boldsymbol{ReLU}_i(\boldsymbol{b})$ coincides with $\boldsymbol{Proj}_i(\boldsymbol{b})$ in the case of $b_i \leqslant 0$.

## 2.2 Polyhedra

In a view of algebra, a polyhedron is the solution space of a (finite) set of linear inequalities $\boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{b}$ where $\boldsymbol{A}$ is a coefficient matrix, $\boldsymbol{x}$ is a vector variable, and $\boldsymbol{b}$ is a constant vector. We call such a kind of representation of polyhedra the H-representation.

Indeed, from the geometry perspective, a polyhedron $\mathcal{X}$ can also be characterized by

$$\mathcal{X} = \left\{ \sum_{i=1}^{k} a_i \boldsymbol{v}_i + \sum_{j=1}^{m} b_j \boldsymbol{d}_j \;\middle|\; a_i, b_j \geqslant 0,\ \sum_i a_i = 1 \right\},$$

where $\boldsymbol{v}_i$s and $\boldsymbol{d}_j$s are given vectors, and each $\boldsymbol{v}_i$ is called a vertex and $\boldsymbol{d}_j$ is a recession direction (or just direction). This is called the V-representation.

For a polyhedron $\mathcal{X}$ in the V-representation, a vertex $\boldsymbol{v}$ is said to be extreme if it is not a convex combination of any other two elements belonging to $\mathcal{X}$. Likewise, a direction is extreme if it cannot be the convex combination of any other directions.

According to the Weyl-Minkowski theorem (for example, see [27]), one can convert interchangeably one representation to the other. Nevertheless, no converting approach that can be adopted in deterministic polynomial time is known so far for either direction.

For a polyhedron $\mathcal{X}$, we say its dimension is $n$, denoted by $\dim \mathcal{X} = n$, if:

1) there exist $n+1$ points $\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ making the set $\{\boldsymbol{v}_i - \boldsymbol{v}_0 \mid i = 1, \ldots, n\}$ linearly independent;

2) every set $\{\boldsymbol{u}_i - \boldsymbol{u}_0 \mid 1 \leqslant i \leqslant m\}$ is linearly dependent provided that $m > n$ and $\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_m \in \mathcal{X}$;

in addition, for a space $\mathcal{X} = \bigcup_{i=1}^{n} \mathcal{X}_i$ where each $\mathcal{X}_i$ is a polyhedron, we define

$$\dim \mathcal{X} = \max_{1 \leqslant i \leqslant n} \dim \mathcal{X}_i.$$

In what follows, we call such a union of finitely many polyhedra a polyhedron bundle (or, simply bundle).

For a bundle, the subspace consisting of all polyhedra with the highest dimension is called the majority part of it.

### 2.3 Recurrent Neural Networks

An RNN consists of one input layer, one output layer, and a sequence of hidden layers. The number of neurons in each layer is called its width. In contrast to FNNs, neurons of a hidden layer of RNNs contain memory units, which retain their previous states when computing the current state, together with the outputs from the previous layer. The operation between two adjacent layers in an RNN is the composition of an affine transformation and a nonlinear activation, such as ReLU, sigmoid, and so on. An example RNN is shown in Fig.1, and besides the input (in blue) and output layers (in green), it also has three hidden layers (in red).

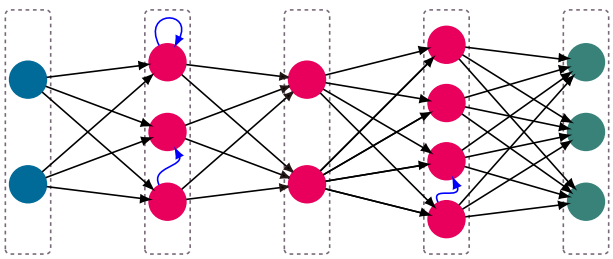

Fig.1. Example RNN.

To model an RNN, we associate each pair of two adjacent layers with three parameters $\boldsymbol{W}$, $\boldsymbol{U}$, and $\boldsymbol{b}$, where $\boldsymbol{W}$ (black arrows in Fig.1) is the connection weight matrix among neurons of two adjacent layers, $\boldsymbol{U}$ (blue arrows in Fig.1) is the memory weight matrix among neurons in the same layer, and $\boldsymbol{b}$ is the bias vector of neurons in each layer. Therefore, an $L$-layer RNN $\mathcal{N}$ can be represented by a sequence of tuples

$$(\boldsymbol{W}_1, \boldsymbol{U}_1, \boldsymbol{b}_1), (\boldsymbol{W}_2, \boldsymbol{U}_2, \boldsymbol{b}_2), \ldots, (\boldsymbol{W}_{L-1}, \boldsymbol{U}_{L-1}, \boldsymbol{b}_{L-1}),$$

where for every $1 \leqslant \ell < L$, $\boldsymbol{W}_\ell$ is the weight matrix between the $\ell$-th layer and the $(\ell+1)$-th layer, and $\boldsymbol{U}_\ell$ and $\boldsymbol{b}_\ell$ are the memory weight matrix and the bias vector of the $(\ell+1)$-th layer, respectively. The input layer generally does not include the memory mechanism or bias vector.

Such an RNN $\mathcal{N}$ corresponds to a function

$$f_{\mathcal{N}} = f_{L-1} \circ \cdots \circ f_2 \circ f_1,$$

where each $f_\ell$ is determined in the following way. For any $k \in \mathbb{N}^+$ and sequence $\boldsymbol{\tau}$, we may obtain a series of vector sequences $\boldsymbol{\tau}_0, \boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_k$ defined as

1) $\boldsymbol{\tau}_0 = \boldsymbol{\tau}$;

2) suppose $\boldsymbol{\tau}_{i-1} = \boldsymbol{c}'_1, \ldots, \boldsymbol{c}'_{i-1}, \boldsymbol{c}_i, \boldsymbol{c}_{i+1}, \ldots, \boldsymbol{c}_k$, then $\boldsymbol{\tau}_i = \boldsymbol{c}'_1, \ldots, \boldsymbol{c}'_{i-1}, \boldsymbol{c}'_i, \boldsymbol{c}_{i+1}, \ldots, \boldsymbol{c}_k$, where

$$\boldsymbol{c}'_i = \boldsymbol{ReLU}(\boldsymbol{W}_\ell \cdot \boldsymbol{c}_i + \boldsymbol{U}_\ell \cdot \boldsymbol{c}'_{i-1} + \boldsymbol{b}_\ell),$$

and we particularly let $\boldsymbol{c}_0 = \boldsymbol{0}$ — intuitively, $\boldsymbol{c}_i$ and $\boldsymbol{c}'_i$ correspond to the input and output of the $\ell$-th layer at time step $i$, respectively;

3) then we let $f_\ell(\boldsymbol{\tau}) = \boldsymbol{\tau}_k$.

When all the memory weight matrices are zero matrices, RNNs degenerate into FNNs.

Therefore, we can unroll the computation process of RNNs in two directions. One direction is the computation from the input layer to the output layer according to the network structure, named spatial direction, and the other direction unrolls the network according to time steps, called temporal direction. Fig.2 illustrates the unrolling of the RNN shown in Fig.1, where the spatial unrolling is presented from left to right, while the temporal unrolling is presented from top to bottom. All the color marks in Fig.2 mean the same as those in Fig.1.

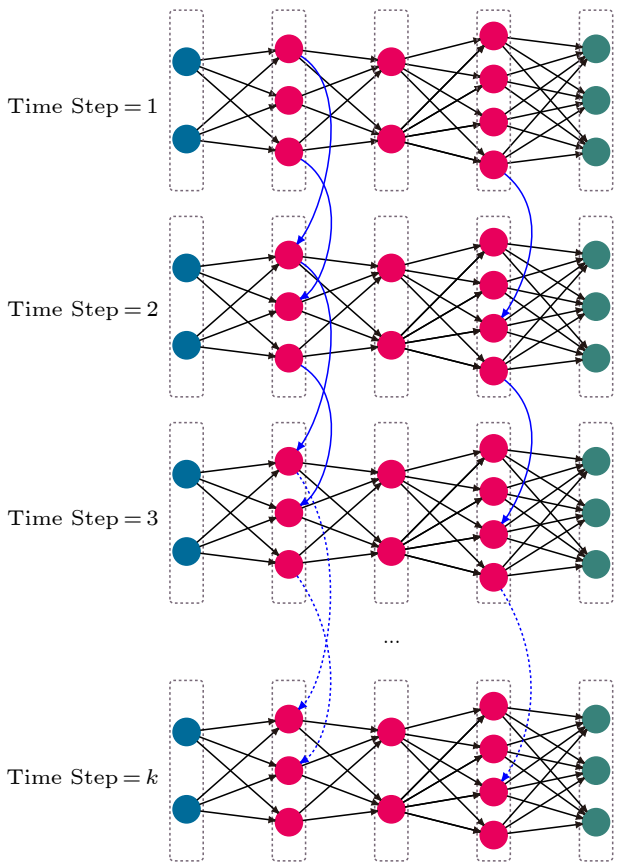


Fig.2. Unrolling illustrations of the RNN in Fig.1.

As a matter of fact, RNNs become FNN-similar networks via unrolling, and however, there lie signifi-

cant differences between the unrolled RNNs and FNNs. For one thing, in view of the network structure, we extend the "loops" from the previous time step to the next one. There still exist the weights connecting the neurons belonging to the same network layer in the unrolled RNNs, as shown in Fig.2. Therefore, the unrolled RNNs just look similar to FNNs and the inherent "loops" have not been broken essentially. For another thing, from the perspective of the computing process, due to the existence of "loops", the computing process in the unrolled RNNs works the same as that in RNNs, i.e., the inputs of the next time step depend on the previous outputs. Thus, it needs iterative computation in the information flow between adjacent layers, which is different from the direct one-step propagation in FNNs and is illustrated in Fig.3.

Therefore, these differences render the FNN-specific verification methods difficult and inapplicable for RNN verification. However, the unrolling process provides us with a global idea of considering the input/output information, or the hidden states. In Section 3, we define the model checking problem on RNNs.

## 3 RNN Model Checking: Problem Definition

### 3.1 Specification Language

To perform model checking, we use an extension $\mathrm{LTL}_f[\boldsymbol{x}]$ of $\mathrm{LTL}_f$ as the specification language, because both the inputs and outputs of an RNN are vector sequences with finite lengths. Meanwhile, since what we are really concerned about are the numeric relations among the data generated during computation, we employ terms to refine the atomic propositions in the base logic. To this end, we use a fixed symbol $\boldsymbol{x}$ to designate the present input/output vector. For example, let $\boldsymbol{\tau} = \boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_k$ be the input sequence, then at the $i$-th step (or moment), $\boldsymbol{x}$ just corresponds to $\boldsymbol{b}_i$, and we further let $\mathsf{X}\boldsymbol{x}$ stand for $\boldsymbol{b}_{i+1}$ for convenience. Formally, we use the following abstract grammar to define terms in such logic:

$$\boldsymbol{t} ::= \boldsymbol{x} \mid \boldsymbol{c} \mid \boldsymbol{M}\boldsymbol{t} \mid \boldsymbol{t} + \boldsymbol{t} \mid \mathsf{X}\boldsymbol{t},$$

where $\boldsymbol{c}$ and $\boldsymbol{M}$ range over (constant) real vectors and real matrices having proper shapes, respectively. In addition, the interpretation of a term $\boldsymbol{t}$ w.r.t. the sequence $\boldsymbol{\tau} = \boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}$ and a position $i$ is given by $[\![\boldsymbol{t}]\!]_{\boldsymbol{\tau},\, i}$. Inductively:

1) $[\![\boldsymbol{x}]\!]_{\boldsymbol{\tau},\, i} = \begin{cases} \boldsymbol{b}_i, & \text{if } 0 \leqslant i < n, \\ 0, & \text{otherwise;} \end{cases}$

2) $[\![\boldsymbol{c}]\!]_{\boldsymbol{\tau},\, i}$ for a constant vector $\boldsymbol{c}$;

3) $[\![\boldsymbol{M}\boldsymbol{t}]\!]_{\boldsymbol{\tau},\, i} = \boldsymbol{M}\,[\![\boldsymbol{t}]\!]_{\boldsymbol{\tau},\, i}$;

4) $[\![\boldsymbol{t}_1 + \boldsymbol{t}_2]\!]_{\boldsymbol{\tau},\, i} = [\![\boldsymbol{t}_1]\!]_{\boldsymbol{\tau},\, i} + [\![\boldsymbol{t}_2]\!]_{\boldsymbol{\tau},\, i}$;

5) $[\![\mathsf{X}\boldsymbol{t}]\!]_{\boldsymbol{\tau},\, i} = [\![\boldsymbol{t}]\!]_{\boldsymbol{\tau},\, i+1}$.

Subsequently, formulas of such $\mathrm{LTL}_f[\boldsymbol{x}]$ extension are defined as:

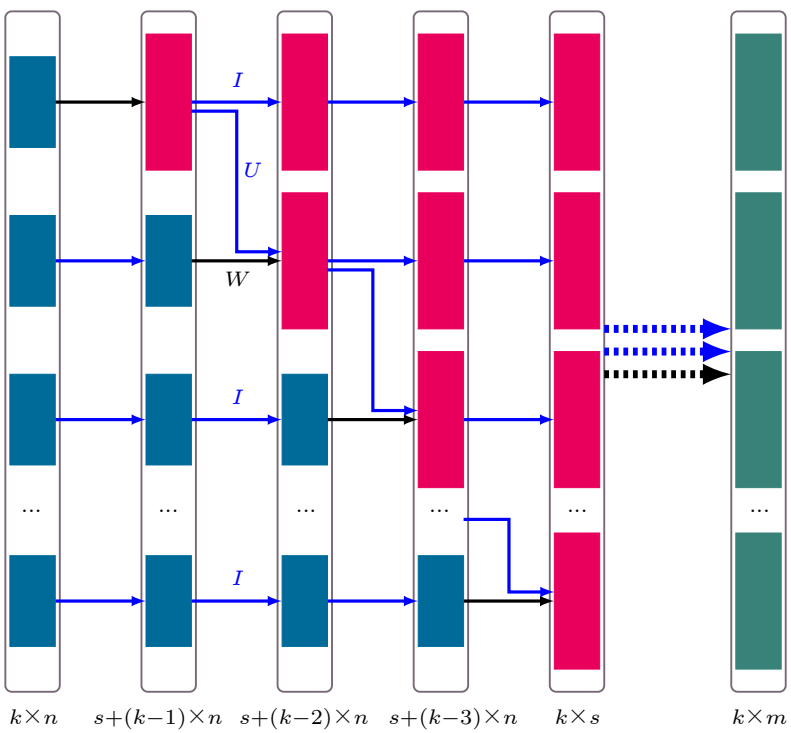


Fig.3. Overview of the polyhedron propagation.

$$\varphi ::= \top \mid \boldsymbol{t}\sim\boldsymbol{t} \mid \neg\varphi \mid \varphi\vee\varphi \mid \mathsf{X}\varphi \mid \varphi\mathsf{U}\varphi,$$

and semantics of such formulas are also given w.r.t. a vector sequence $\boldsymbol{\tau}$ and a position $i$, namely that:

1) $\boldsymbol{\tau},\ i\models\top$ trivially holds;

2) $\boldsymbol{\tau},\ i\models\boldsymbol{t}_1\sim\boldsymbol{t}_2$ if and only if $[\![\boldsymbol{t}_1]\!]_{\boldsymbol{\tau},\ i}\sim[\![\boldsymbol{t}_2]\!]_{\boldsymbol{\tau},\ i}$;

3) $\boldsymbol{\tau},\ i\models\neg\varphi$ if and only if $\boldsymbol{\tau},\ i\not\models\varphi$;

4) $\boldsymbol{\tau},\ i\models\varphi_1\vee\varphi_2$ if and only if $\boldsymbol{\tau},\ i\models\varphi_1$ or $\boldsymbol{\tau},\ i\models\varphi_2$;

5) $\boldsymbol{\tau},\ i\models\mathsf{X}\varphi$ if and only if $\boldsymbol{\tau},\ i+1\models\varphi$;

6) $\boldsymbol{\tau},\ i\models\varphi_1\mathsf{U}\varphi_2$ if and only if there is some $k<\mathrm{len}(\boldsymbol{\tau})$ such that $\boldsymbol{\tau},k\models\varphi_2$ and $\boldsymbol{\tau},j\models\varphi_1$ for each $j$ with $i\leqslant j<k$.

In addition, we directly write $\boldsymbol{\tau}\models\varphi$ in the case of $i=0$.

We define derived Boolean connectives such as $\wedge$, $\rightarrow$ as usual, and also define the following derived temporal connectives for convenience:

1) $\mathsf{F}\varphi\overset{\text{def}}{=}\top\mathsf{U}\varphi$;

2) $\mathsf{G}\varphi\overset{\text{def}}{=}\neg\mathsf{F}\neg\varphi$;

3) $\varphi_1\mathsf{R}\varphi_2\overset{\text{def}}{=}\neg(\neg\varphi_1\mathsf{U}\neg\varphi_2)$.

For any given $n$ and $k$, let $[\![\varphi]\!]_{n,\ k}$ be the set $\{\boldsymbol{\tau}\in(\mathbb{R}^n)^k\mid\boldsymbol{\tau}\models\varphi\}$.

Remind that in this logic, the operator $\mathsf{X}$ acts as both a function and a connective. Indeed, $\mathsf{X}$ is communicative and distributive with other operators, namely:

1) $[\![\mathsf{X}\boldsymbol{M}\boldsymbol{t}]\!]_{\boldsymbol{\tau},\ i}=[\![\boldsymbol{M}\mathsf{X}\boldsymbol{t}]\!]_{\tau,\ i}$;

2) $[\![\mathsf{X}(\boldsymbol{t}_1+\boldsymbol{t}_2)]\!]_{\boldsymbol{\tau},\ i}=[\![\mathsf{X}(\boldsymbol{t}_1)+\mathsf{X}(\boldsymbol{t}_2)]\!]_{\boldsymbol{\tau},\ i}$;

3) $[\![\mathsf{X}(\boldsymbol{t}_1\sim\boldsymbol{t}_2)]\!]_{n,\ k}=[\![\mathsf{X}\boldsymbol{t}_1\sim\mathsf{X}\boldsymbol{t}_2]\!]_{n,\ k}$;

4) $[\![\mathsf{X}\neg\varphi]\!]_{n,\ k}=[\![\neg\mathsf{X}\varphi]\!]_{n,\ k}$;

5) $[\![\mathsf{X}(\varphi_1\vee\varphi_2)]\!]_{n,\ k}=[\![\mathsf{X}\varphi_1\vee\mathsf{X}\varphi_2]\!]_{n,\ k}$;

6) $[\![\mathsf{X}(\varphi_1\mathsf{U}\varphi_2)]\!]_{n,\ k}=[\![(\mathsf{X}\varphi_1)\mathsf{U}(\mathsf{X}\varphi_2)]\!]_{n,\ k}$.

**Theorem 2.** *$[\![\varphi]\!]_{n,\ k}$ composes a bundle in $\mathbb{R}^{n\times k}$.*

*Proof.* We define a translator $\mathscr{T}$ which eliminates all temporal connectives in $\varphi$ and equivalently transform it in a Boolean combination of inequalities w.r.t. some variable $\boldsymbol{y}\in\mathbb{R}^{n\times k}$. Inductively, for a position $i$:

1) $\mathscr{T}(\boldsymbol{c},i)=\boldsymbol{c}$;

2) $\mathscr{T}(\boldsymbol{x},i)=\begin{cases}\left[0_{in,\ n}\vdots\boldsymbol{I}_n\vdots0_{(k-1)n,\ n}\right]\boldsymbol{y}, & \text{if } i<k,\\ 0_{n\times1}, & \text{if } i\geqslant k,\end{cases}$

where $\boldsymbol{I}_n$ is the identity matrix with size $n$ by $n$, whereas $0_{m,\ n}$ is the zero matrix with size $m$ by $n$;

3) $\mathscr{T}(\boldsymbol{M}\boldsymbol{t},i)=\boldsymbol{M}\mathscr{T}(\boldsymbol{t},i)$;

4) $\mathscr{T}(\boldsymbol{t}_1+\boldsymbol{t}_2,i)=\mathscr{T}(\boldsymbol{t}_1,i)+\mathscr{T}(\boldsymbol{t}_2,i)$;

5) $\mathscr{T}(\mathsf{X}\boldsymbol{t},i)=\mathscr{T}(\boldsymbol{t},i+1)$;

6) $\mathscr{T}(\boldsymbol{t}_1\sim\boldsymbol{t}_2,i)=\mathscr{T}(\boldsymbol{t}_1,i)\sim\mathscr{T}(\boldsymbol{t}_2,i)$;

7) $\mathscr{T}(\neg\psi,i)=\neg\mathscr{T}(\psi,i)$;

8) $\mathscr{T}(\varphi_1\vee\varphi_2,i)=\mathscr{T}(\varphi_1,i)\vee\mathscr{T}(\varphi_2,i)$;

9) $\mathscr{T}(\mathsf{X}\psi,i)=\mathscr{T}(\psi,i+1)$;

10) $\mathscr{T}(\varphi_1\mathsf{U}\varphi_2,i)=\bigvee_{j=i}^{k-1}\left(\mathscr{T}(\varphi_2,j)\wedge\bigwedge_{t=i}^{j-1}\mathscr{T}(\varphi_1,t)\right)$.

Then, we just let $\mathscr{T}(\varphi,0)$ be the resulting description, which corresponds to a bundle within $\mathbb{R}^{n\times k}$. Indeed, one can show case by case that $\boldsymbol{\tau},\ i\models\varphi$ if and only if $\mathrm{Jux}(\boldsymbol{\tau})$ is within the solution space of $\mathscr{T}(\varphi,i)$. □

### 3.2 Qualitative and Quantitative Model Checking

Given an RNN $\mathcal{N}$ with the input width $n$ and the output width $m$, let $f_{\mathcal{N}}$ be the corresponding function determined by $\mathcal{N}$; given two $\mathrm{LTL}_f[\boldsymbol{x}]$ formulas $\varphi$ and $\psi$, a parameter $k\in\mathbb{N}$ of sequence length, then:

1) the goal of qualitative model checking is to check whether $f_{\mathcal{N}}([\![\varphi]\!]_{n,\ k}))\subseteq[\![\psi]\!]_{m,\ k}$ holds;

2) the task of quantitative model checking is to compute the ratio

$$\frac{\mathrm{vol}(f_{\mathcal{N}}([\![\varphi]\!]_{n,\ k})\cap[\![\psi]\!]_{m,\ k})}{\mathrm{vol}(f_{\mathcal{N}}([\![\varphi]\!]_{n,\ k}))},$$

where $\mathrm{vol}(\mathcal{X})$ is the volume of the bundle $\mathcal{X}$, which is explained below.

We use $(\varphi\{\mathcal{N}\}\psi)_k$ to denote the model checking result. For the qualitative case, it is a Boolean value, whereas for the quantitative case, it is some probability.

To define the volume of a bundle, we begin with the bounded case. According to the definition, a bounded bundle can be partitioned into finitely many disjoint bounded polyhedra. Within the Hilbert space $\mathbb{R}^n$, a polyhedron that has exactly $n+1$ extreme vertices is called a simplex. Suppose the simplex $\mathcal{S}$ is determined via a set of vertices $V=\{\boldsymbol{v}_0,\boldsymbol{v}_1,\ldots,\boldsymbol{v}_n\}$ where $\boldsymbol{v}_i=(v_{i,\ 1},v_{i,\ 2},\ldots,v_{i,\ n})^{\mathrm{T}}$. Then, we have $\mathrm{vol}(\mathcal{S})=(1/n!)\mathrm{abs}(\det V)$[28], where $\det V$ is

$$\det V=\det\begin{pmatrix}\boldsymbol{v}_1-\boldsymbol{v}_0\\ \boldsymbol{v}_2-\boldsymbol{v}_0\\ \cdots\\ \boldsymbol{v}_n-\boldsymbol{v}_0\end{pmatrix}=$$

$$\begin{vmatrix}v_{1,\ 1}-v_{0,\ 1} & \cdots & v_{1,\ n}-v_{0,\ n}\\ v_{2,\ 1}-v_{0,\ 1} & \cdots & v_{2,\ n}-v_{0,\ n}\\ \vdots & \vdots & \vdots\\ v_{n,\ 1}-v_{0,\ 1} & \cdots & v_{n,\ n}-v_{0,\ n}\end{vmatrix}=\begin{vmatrix}1 & v_{0,\ 1} & \cdots & v_{0,\ n}\\ 1 & v_{1,\ 1} & \cdots & v_{1,\ n}\\ \vdots & \vdots & \ddots & \vdots\\ 1 & v_{n,\ 1} & \cdots & v_{n,\ n}\end{vmatrix}.$$

Subsequently, for a polyhedron $\mathcal{X}$ consisting of

more than $n+1$ extreme vertices, we need to partition it into a (finite) set of simplexes. Suppose a polyhedron $\mathcal{X}$ is built up upon an extreme vertex set $\hat{V}=\{\boldsymbol{v}_0,\boldsymbol{v}_1,\ldots,\boldsymbol{v}_t\}$ where $t>n+1$. First of all, we fix the vertex $\boldsymbol{v}_0$ and choose other $n$ vertices to compose a simplex. The issue is that we must guarantee that "all the remaining vertices are located on the same side of the hyperplane determined by the $n$ chosen vertices". In this case, we say that these $n$ chosen vertices constitute a promising subset of $\hat{V}$. Formally, suppose that $V'=\{\boldsymbol{v}_{i_1},\ldots,\boldsymbol{v}_{i_n}\}\subset\hat{V}$, then the function of the hyperplane containing $V'$ is

$$P_{V'}(\boldsymbol{x})=\begin{vmatrix}1 & x_1 & x_2 & \ldots & x_n\\ 1 & v_{i_1,\,1} & v_{i_1,\,2} & \ldots & v_{i_1,\,n}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1 & v_{i_n,\,1} & v_{i_n,\,2} & \ldots & v_{i_n,\,n}\end{vmatrix}=0.$$

Consequently, $V'$ is promising iff

$$P_{V'}(\boldsymbol{v}_0)\times P_{V'}(\boldsymbol{v})\geqslant 0,$$

for each $\boldsymbol{v}\in\hat{V}\setminus V'$. With $\mathrm{prom}(V)$ denoting all promising subsets of $V$, we then have

$$\mathrm{vol}(\mathcal{X})=\sum_{V'\in\mathrm{prom}(V)}\frac{\mathrm{abs}(P_{V'}(\boldsymbol{v}_0))}{n!}. \tag{1}$$

Lastly, for an unbounded polyhedron $\mathcal{X}$, we need to temporarily compute the volume of the intersection of $\mathcal{X}$ and the polyhedron

$$-M\leqslant x_i\leqslant M,\qquad i=1,\,2,\,\ldots,\,n,$$

(the intersection is denoted by $\mathcal{X}_M$) using the aforementioned approach. Here, we need to treat $M$ as a symbol, rather than a concrete value. As a result, what we get is a polynomial about $M$. From (1), we obtain two polynomials:

$$h_{\mathcal{X}}(M)=\sum_{i=0}^{n}a_iM^i,\qquad h_{\mathcal{Y}}(M)=\sum_{j=0}^{m}b_jM^j,$$

for $\mathcal{X}_M$ and $\mathcal{Y}_M$, respectively. Then, by definition, we have

$$\frac{\mathrm{vol}(\mathcal{X})}{\mathrm{vol}(\mathcal{Y})}=\lim_{M\to\infty}\frac{\sum_{i=0}^{n}a_iM^i}{\sum_{j=0}^{m}b_jM^j}=\begin{cases}0, & n<m,\\ \dfrac{a_n}{b_m}, & n=m,\\ \infty, & n>m.\end{cases}$$

Since the technique computing the volume of unbounded bundles requires viewing parameter $M$ as a symbol, it is called the symbolic approach.

## 4 Verification Framework

We put the emphasis on the quantitative model checking algorithm. The approach is mainly based on the so-called "polyhedron forward propagation". Given an RNN

$$\mathcal{N}=(\boldsymbol{W}_1,\boldsymbol{U}_1,\boldsymbol{b}_1),(\boldsymbol{W}_2,\boldsymbol{U}_2,\boldsymbol{b}_2),\ldots,(\boldsymbol{W}_{L-1},\boldsymbol{U}_{L-1},\boldsymbol{b}_{L-1}),$$

(with the input/output width $n$ and $m$), pre-condition $\varphi$, post-condition $\psi$, and length bound $k$, ideally, the process computes a series of bundles $\mathcal{X}_0,\mathcal{X}_1,\ldots,\mathcal{X}_{L-1}$, where $\mathcal{X}_0=[\![\varphi]\!]_{n,\,k}$ and $\mathcal{X}_{L-1}=f_{\mathcal{N}}(\mathcal{X}_0)$. To accomplish this, we also need to produce a series of intermediate bundles

$$\mathcal{X}_{\ell,\,0},\mathcal{X}_{\ell,\,1},\ldots,\mathcal{X}_{\ell,\,k},$$

to obtain $\mathcal{X}_{\ell+1}$ from $\mathcal{X}_\ell$, where $\mathcal{X}_{\ell,\,0}=\mathcal{X}_\ell$ and

$$\mathcal{X}_{\ell,\,i+1}=\boldsymbol{ReLU}_{X_i}(\boldsymbol{A}_{\ell,\,i}\cdot\mathcal{X}_{\ell,\,i}+\boldsymbol{d}_{\ell,\,i}), \tag{2}$$

and $\mathcal{X}_{\ell+1}$ is just $\mathcal{X}_{\ell,\,k}$. In (2), components are given as below.

1) $\boldsymbol{A}_{\ell,\,i}=\begin{pmatrix}\boldsymbol{I}_{n_{\ell+1}\times(i-1)} & & & \\ & \boldsymbol{I}_{n_{\ell+1}} & & \\ & \boldsymbol{U}_\ell & \boldsymbol{W}_\ell & \\ & & & \boldsymbol{I}_{n_\ell\times(k-i-1)}\end{pmatrix}$

for each $i\geqslant 1$; and particularly,

$$\boldsymbol{A}_{\ell,\,0}=\begin{pmatrix}\boldsymbol{W}_\ell & \\ & \boldsymbol{I}_{n_\ell\times(k-1)}\end{pmatrix}.$$

Here, $n_\ell$ is the width of $\mathcal{N}$'s $\ell$-th layer.

2) $\boldsymbol{d}_{\ell,\,i}=(0^{\mathrm{T}}_{n_{\ell+1}\times i},\boldsymbol{b}_\ell^{\mathrm{T}},0^{\mathrm{T}}_{n_\ell\times(k-i-1)})^{\mathrm{T}}$, here $\boldsymbol{0}_m$ is the zero vector with length $m$.

3) The index set $X_i=\{i\cdot n_{\ell+1}+j\mid 1\leqslant j\leqslant n_{\ell+1}\}$.

Fig.3 illustrates the above polyhedron propagation figuratively, where the arrows in black, blue, and blue represent the matrices $\boldsymbol{W}$, $\boldsymbol{U}$, and $\boldsymbol{I}$, respectively. A block stands for the whole network layer in Fig.1 in the same color and only the first hidden layer is displayed here. Without loss of generality, we assume the dimensions of the input, output, and the first hidden layer to be $n$, $m$, and $s$, respectively, meeting the formalization need.

The above process is rigorous, and unfortunately, we will encounter the "vertex number explosion" problem. Let us see how it happens. Just consider a polyhedron $\mathcal{X}$ with $v$ vertices and $d$ directions ($v>0$ and $d\geqslant 0$), and since the number of vertices and directions will not increase after an affine transforma-

tion, we only need to focus on the ReLU operation. Since we always have

$$\boldsymbol{ReLU}_X = \boldsymbol{ReLU}_{i_1} \circ \boldsymbol{ReLU}_{i_2} \circ \cdots \circ \boldsymbol{ReLU}_{i_r},$$

provided that $X = \{i_1, i_2, \ldots, i_r\}$, we thus just need to study the ReLU operation w.r.t. a single coordinate, i.e., $\boldsymbol{ReLU}_i$. It can be observed that ReLU does not preserve convexity, which means that one polyhedron might be transformed into a bundle[②].

It is notable that for a polyhedron, the number of extreme directions is bounded by its dimension, and the number does not exceed the maximum network width multiplying the length of input sequences. In contrast, suppose the numbers of vertices having non-negative and negative $i$-th coordinates in $\mathcal{X}$ are $d_1$ and $d_2$, respectively; then we will have $O(d_1 \times d_2)$ vertices in the bundle $\boldsymbol{ReLU}_i(\mathcal{X})$ in total.

For this reason, we need to do some abstraction (more accurately, approximation) of the intermediate bundles to reduce the vertex number. In addition, for every polyhedron member $\mathcal{X}$ in the bundle, its abstraction $\mathcal{X}'$ must fulfill the following requirements.

1) $\mathcal{X}' \supseteq \mathcal{X}$;

2) $\dim \mathcal{X}' = \dim \mathcal{X}$;

3) $\mathcal{X}' \setminus \mathcal{X}$ is bounded.

To achieve this, we need two auxiliary algorithms.

1) The first is used to compute $\boldsymbol{ReLU}_i(\mathcal{X})$ from a polyhedron $\mathcal{X}$.

2) The second is related to the polyhedron abstraction, which preserves the dimension, but reduces the number of vertices.

*Computing $\boldsymbol{ReLU}_i(\mathcal{X})$ from Polyhedron $\mathcal{X}$.* The point of this algorithm is to avoid the inter-conversion between H- and V-representations. An algorithm that can be done in polynomial time is given below. Suppose that $\mathcal{X}$ is characterized by the extreme vertex set $V$ and the extreme direction set $R$. Let

$$\boldsymbol{e}_i = (\underbrace{0, \ldots, 0}_{i-1}, 1, 0, \ldots, 0)^{\mathrm{T}},$$

and let $\mathcal{H}_i$ be the coordinate plane $\{\boldsymbol{x} \mid \boldsymbol{x}^{\mathrm{T}}\boldsymbol{e}_i = 0\}$ (i.e., the set consisting of points whose $i$-th coordinate is 0). One should be aware that $\boldsymbol{ReLU}_i(\mathcal{X})$ may contain at most two polyhedra — the one "above" $\mathcal{H}_i$ (denoted by $\mathcal{X}_0$) and the one "within" $\mathcal{H}_i$ (denoted by $\mathcal{X}_1$).

1) First of all, the vertex sets of the resultant polyhedra can be determined as follows. Let

$$\begin{aligned} V' = \{\boldsymbol{v} \in \mathcal{H}_i \mid \exists \boldsymbol{v}_1, \boldsymbol{v}_2 \in V, \exists 1 \geqslant c \geqslant 0,\\ \text{s.t. } \boldsymbol{v} = c\boldsymbol{v}_1 + (1-c)\boldsymbol{v}_2\}\\ \cup \{\boldsymbol{v} \in \mathcal{H}_i \mid \exists \boldsymbol{v}' \in V, \exists \boldsymbol{d} \in R, \exists c \geqslant 0\\ \text{s.t. } \boldsymbol{v} = \boldsymbol{v}' + c\boldsymbol{d}\}, \end{aligned}$$

and then the vertex set of $\mathcal{X}_0$ is $\{\boldsymbol{v} \in V \mid \boldsymbol{v}^{\mathrm{T}}\boldsymbol{e}_i \geqslant 0\} \cup V'$, and that of $\mathcal{X}_1$ is $\{\boldsymbol{ReLU}_i(\boldsymbol{v}) \mid \boldsymbol{v} \in V, \boldsymbol{v}^{\mathrm{T}}\boldsymbol{e}_i \leqslant 0\} \cup V'$.

2) Second, we establish the direction sets for $\mathcal{X}_0$ and $\mathcal{X}_1$. For two directions $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ with $\boldsymbol{d}_1^{\mathrm{T}} \cdot \boldsymbol{e}_i < 0$ and $\boldsymbol{d}_2^{\mathrm{T}} \cdot \boldsymbol{e}_i > 0$, we let

$$\mathrm{comb}(\boldsymbol{d}_1, \boldsymbol{d}_2) = (\boldsymbol{d}_2^{\mathrm{T}} \cdot \boldsymbol{e}_i)\boldsymbol{d}_1^{\mathrm{T}} - (\boldsymbol{d}_1^{\mathrm{T}} \cdot \boldsymbol{e}_i)\boldsymbol{d}_2^{\mathrm{T}},$$

and let $R' = \{\mathrm{comb}(\boldsymbol{d}_1, \boldsymbol{d}_2) \mid \boldsymbol{d}_1, \boldsymbol{d}_2 \in R, \boldsymbol{d}_1^{\mathrm{T}} \cdot \boldsymbol{e}_i < 0, \text{ and } \boldsymbol{d}_2^{\mathrm{T}} \cdot \boldsymbol{e}_i > 0\}$, and then the direction set of $\mathcal{X}_0$ is $R' \cup \{\boldsymbol{d} \in R \mid \boldsymbol{d}^{\mathrm{T}} \cdot \boldsymbol{e}_i \geqslant 0\}$ and the direction set of $\mathcal{X}_1$ is $R' \cup \{\boldsymbol{ReLU}(\boldsymbol{d}) \mid \boldsymbol{d} \in R, \boldsymbol{d}^{\mathrm{T}} \cdot \boldsymbol{e}_i \leqslant 0\}$.

Remind that $\mathcal{X}_0$ (and/or $\mathcal{X}_1$) might be $\emptyset$ if its vertex set is empty. We illustrate the algorithm upon 2-dimensional example polyhedra in Fig.4 and Fig.5, which are with respect to the bounded and unbounded cases, respectively. Fig.4 shows the detailed algorithm process on each dimension, while Fig.5 only shows the processing results on the dimensions.

**Theorem 3.** *$\{\mathcal{X}_0, \mathcal{X}_1\}$ is precisely the bundle of $\boldsymbol{ReLU}_i(\mathcal{X})$.*

*Proof.* First, we have that $\mathcal{X} = \mathcal{X}_{\geqslant 0} \cup \mathcal{X}_{\leqslant 0}$, where $\mathcal{X}_{\geqslant 0}$ and $\mathcal{X}_{\leqslant 0}$ are the intersections of $\mathcal{X}$ with the polyhedra determined by $x_i \geqslant 0$ and $x_i \leqslant 0$, respectively. Thus, $\boldsymbol{ReLU}_i(\mathcal{X}) = \boldsymbol{ReLU}_i(\mathcal{X}_{\geqslant 0}) \cup \boldsymbol{ReLU}_i(\mathcal{X}_{\leqslant 0})$. For both $\mathcal{X}_{\geqslant 0}$ and $\mathcal{X}_{\leqslant 0}$, the operation $\boldsymbol{ReLU}_i$ is linear. In addition, we definitely have $\boldsymbol{ReLU}_i(\mathcal{X}_{\geqslant 0}) = \mathcal{X}_{\geqslant 0}$ and $\boldsymbol{ReLU}_i(\mathcal{X}_{\leqslant 0}) = \boldsymbol{Proj}_i(\mathcal{X}_{\leqslant 0})$.

The vertice set of $\mathcal{X}_{\geqslant 0}$ can be categorized into two parts.

1) The first part contains the vertices $\boldsymbol{v}$ of $\mathcal{X}$ such that $\boldsymbol{e}_i^{\mathrm{T}} \cdot \boldsymbol{v} \geqslant 0$;

2) the other part is constituted with vertices of the intersection of $\mathcal{X}$ and the plane $x_i = 0$.

Meanwhile, an extreme direction $\boldsymbol{d}$ of $\mathcal{X}_{\geqslant 0}$ must be either of the following two cases.

1) It is an (extreme) direction of $\mathcal{X}$, and $\boldsymbol{e}_i^{\mathrm{T}} \cdot \boldsymbol{d} \geqslant 0$;

2) or it is located in the newly generated face $x_i = 0$.

[②]Actually, from the perspective of topology[29], such a bundle must be a "complex" — it consists of a set of simplexes, and the common part of each adjacent two is a simplex with lower dimension.

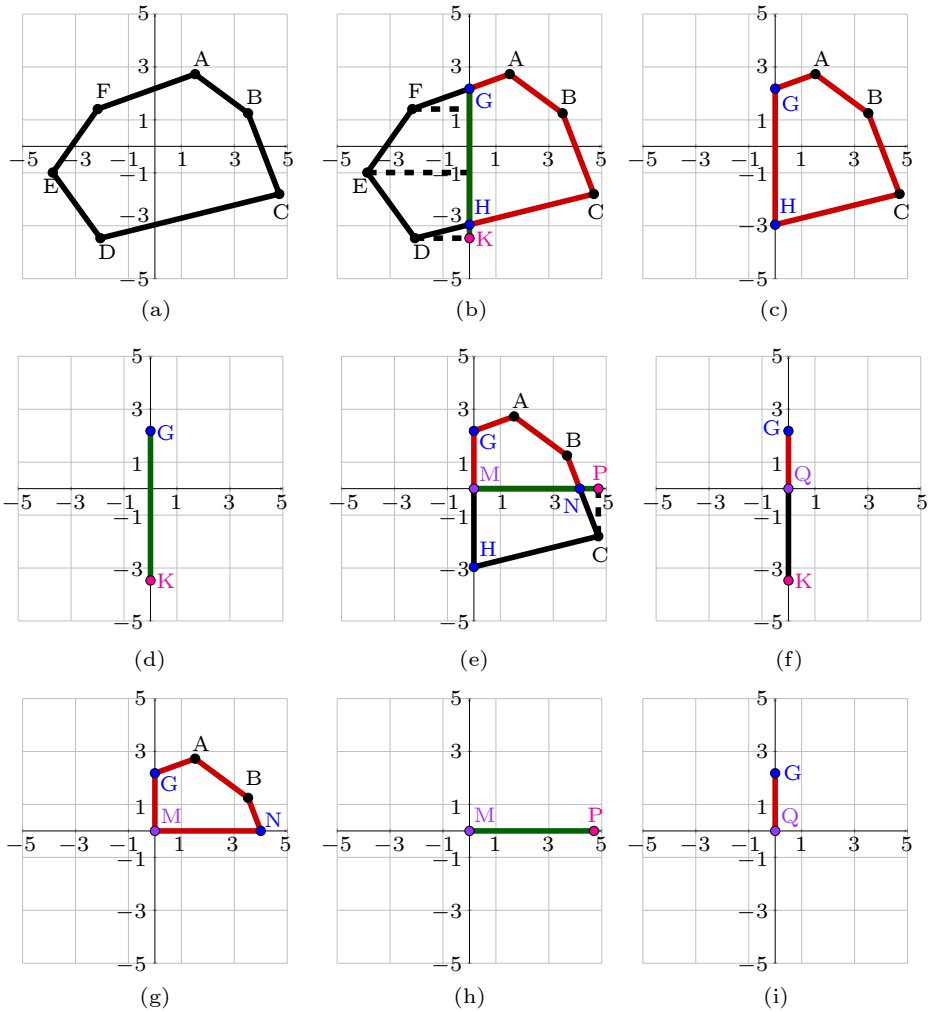


Fig.4. Process of the ReLU algorithm on a 2D example (bounded). (a) Original polyhedron. (b) Proceeding w.r.t. the 1st coordinate. (c)–(d) Intermediate results. (e)–(f) Proceeding w.r.t. the 2nd coordinate. (g)–(i) Polyhedron bundle of the final results.

For both cases, one can check that $\mathcal{X}_{\geqslant 0}$ coincides with $\mathcal{X}_0$ — though during the construction, we may introduce some non-extreme vertices and directions.

Likewise, we can also see that $\mathcal{X}_1$ is precisely $\boldsymbol{ReLU}_i(\mathcal{X}_{\leqslant 0})$. We thus have $\boldsymbol{ReLU}_i(\mathcal{X}) = \{\mathcal{X}_0, \mathcal{X}_1\}$. □

*Polyhedron Abstraction.* Indeed, within $\mathbb{R}^n$, for a given polyhedron $\mathcal{X} \subseteq \mathbb{R}^n$ with $\dim \mathcal{X} = m$ and $m < n$, we can use a rigid linear deformation $\boldsymbol{R}$ to transform it into the space $\mathbb{R}^m$ in the following way.

Suppose that we have $m+1$ points $\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_m \in \mathcal{X}$ and the set $\{\boldsymbol{p}_i - \boldsymbol{p}_0\}_{1 \leqslant i \leqslant m}$ is linearly independent, and then let $\boldsymbol{v}_i = \boldsymbol{p}_i - \boldsymbol{p}_0$, let $\{\boldsymbol{u}_i\}_{1 \leqslant i \leqslant m}$ be the orthogonal basis obtained from $\{\boldsymbol{v}_i\}_{1 \leqslant i \leqslant m}$ via applying Gram-Schmdit orthogonalization. Then just let

$$\boldsymbol{R} : \boldsymbol{v} \mapsto (c_1, \ldots, c_m)^{\mathrm{T}},$$

where $c_i = (\boldsymbol{v} - \boldsymbol{p}_0)^{\mathrm{T}} \boldsymbol{u}_i$. The following claims are straightforward to check.

**Theorem 4.** *For the linear transformation $\boldsymbol{R}$ we have:*

1) *it is a bijection from $\mathcal{X}$ to $\boldsymbol{R}(\mathcal{X})$;*

2) *$\boldsymbol{R}$ is rigid, namely, we have* $\mathrm{vol}(\boldsymbol{R}(\mathcal{X})) = \mathrm{vol}(\mathcal{X})$.

In addition, the inversion of $\boldsymbol{R}$ is given by $\boldsymbol{R}^{-1}(\boldsymbol{t}) = \boldsymbol{U}\boldsymbol{t} + \boldsymbol{p}_0$, where $\boldsymbol{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m)$.

The abstraction algorithm is an extension of Yu's work presented in [30] (Algorithm 1 in [30], hereinafter referring it as AoYu), which is an iterative truncation procedure, starting from an initial over-approximated polyhedron. During each iteration, it selects a cutting hyperplane (c.f. lines 2–4, 11–13 of AoYu) and computes the new polyhedron after truncation (c.f. lines 7–10 of AoYu). Given an $m$-dimensional polyhedron $\mathcal{X}$, with the extreme vertex set $V$ and direction set $R$, we first transform it into $\mathbb{R}^m$ using the aforementioned rigid deformation, and then the followings are performed.

1) To initialize the $m$-simplex, still let $\boldsymbol{e}_i =$

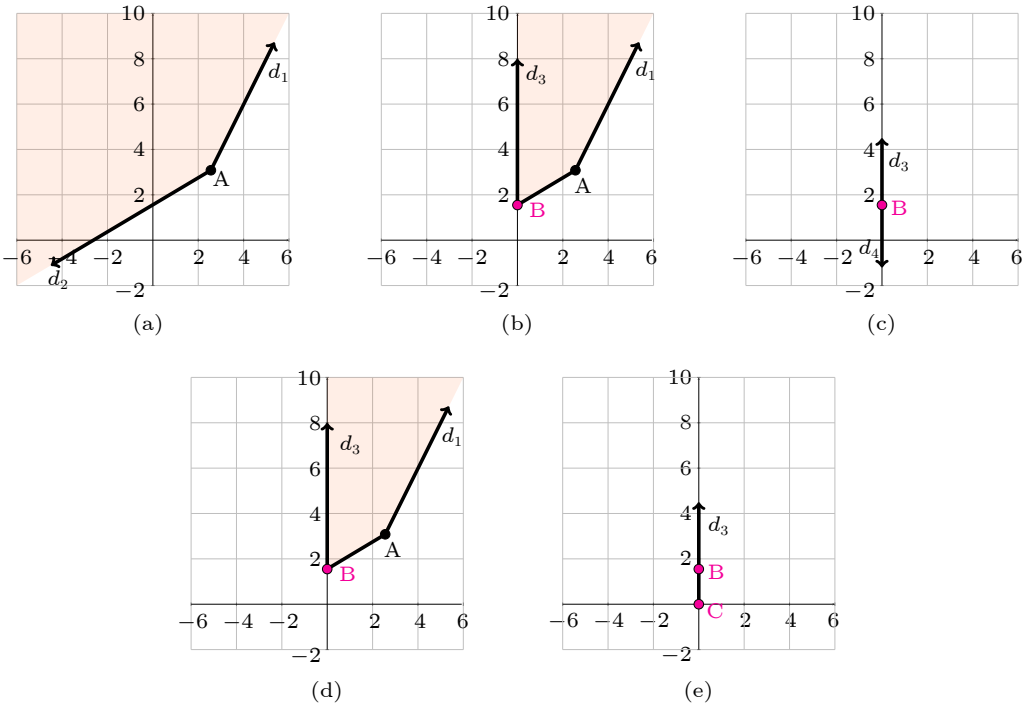


Fig.5. Process of the ReLU algorithm on a 2D example (unbounded). (a) Original polyhedron. (b)–(c) Results of proceeding w.r.t. the 1st coordinate. (d)–(e) Results of proceeding w.r.t. the 2nd coordinate (i.e., final results).

$(\underbrace{0,\cdots,0}_{i-1},1,\underbrace{0,\cdots,0}_{m-i-1})$, let $\boldsymbol{v}_\mu = (\min_{\boldsymbol{v}\in V} \boldsymbol{e}_1^{\mathrm{T}}\cdot\boldsymbol{v},\cdots,\min_{\boldsymbol{v}\in V}\boldsymbol{e}_m^{\mathrm{T}}\cdot\boldsymbol{v})^{\mathrm{T}}$, and then let $\mathcal{X}'=\bigcap_{i=1}^{m+1}\mathcal{H}_i$ where

$$\mathcal{H}_i=\begin{cases}\{\boldsymbol{x}\mid(\boldsymbol{x}-\boldsymbol{v}_\mu)^{\mathrm{T}}\boldsymbol{e_i}\leqslant 0\}, & 0\leqslant i\leqslant m,\\ \{\boldsymbol{x}\mid(\boldsymbol{x}-\boldsymbol{v}_\mu)^{\mathrm{T}}\boldsymbol{s}\leqslant c\}, & i=m+1,\end{cases}$$

and $\boldsymbol{s},c$ can be obtained by solving the optimization problem:

$$\boldsymbol{s},c=\underset{\boldsymbol{t}=(t_1,\ \cdots,\ t_m),\ b}{\arg\max}\ \#\{\boldsymbol{p}\in V\mid(\boldsymbol{p}-\boldsymbol{v}_\mu)^{\mathrm{T}}\boldsymbol{t}=b\}$$
$$\text{s.t.}\quad\begin{cases}(\boldsymbol{v}-\boldsymbol{v}_\mu)^{\mathrm{T}}\boldsymbol{t}\leqslant b, & \boldsymbol{v}\in V,\\ t_i\geqslant 0, & i=1,\ldots,m,\\ b\geqslant 0.\end{cases}$$

2) We update the vertices of the initial $m$-simplex with

$$V'=\mathrm{cvx}(\{\boldsymbol{v}_i+a\boldsymbol{d}_j\mid\boldsymbol{v}_i\in V,\boldsymbol{d}_j\in R,a\in\mathbb{R}\}\cap\mathcal{X}'),$$

to deal with unbounded ones. The $\mathrm{cvx}(S)$ function returns the vertex set of the convex hull of $S$ with the QuickHull algorithm[31].

3) Each iteration will increase the number of vertices. Finally, we replace the loop condition with a bound of the (designated) vertex number or the approximation precision.

The above abstraction algorithm guarantees Requirements 1–3 mentioned in Section 4. Requirement 1 is obvious and requirement 2 is guaranteed by the rigid transformation. To see why requirement 3 holds, we need to observe the following fact: let $\mathcal{X}'$ be the abstracted polyhedron and then for each ray $\mathcal{R}=(\boldsymbol{v},\boldsymbol{d})$[③], according to the construction, there must exist some $\boldsymbol{v}'=\boldsymbol{v}+b\cdot\boldsymbol{d}\in\mathcal{X}$ for some $b\in\mathbb{R}$. Since $\boldsymbol{d}$ is also a recession direction of $\mathcal{X}$, we can declare that $\mathcal{X}\cap\mathcal{R}\neq\emptyset$ — more accurately, $\mathcal{R}\setminus\mathcal{X}$ is a bounded line (a.k.a., a segment). It implies that $\mathcal{X}'\setminus\mathcal{X}$ is bounded — otherwise, there must exist some ray in $\mathcal{X}'\setminus\mathcal{X}$, which is unbounded.

Notably, what we have yielded is a polynomial time algorithm. Fig.6 demonstrates the different stages of the algorithm upon a 2D unbounded polyhedron. Therefore, the propagation upon $\mathcal{N}$ with abstraction also determines a mapping between bundles. We denote the mapping by $f^\sharp_{\mathcal{N}}$ in what follows.

*A Speculative Optimization Approach.* Recall that a bundle might consist of polyhedra with different dimensions, and we have that $\mathrm{vol}(\mathcal{X})/\mathrm{vol}(\mathcal{Y})=0$ from Measure Theory if $\dim\mathcal{X}<\dim\mathcal{Y}$. Also, it can be seen that the dimension of a bundle cannot increase during the propagation — it can sometimes decrease if it is applied to a degenerated (or, singular) linear trans-

[③]Here, we use a tuple $(\boldsymbol{v},\boldsymbol{d})$ to denote a ray within $\mathcal{X}'$, which means $\boldsymbol{v}\in\mathcal{X}'$ and $\boldsymbol{d}$ is a recession direction. It corresponds to the set $\{\boldsymbol{v}+a\cdot\boldsymbol{d}\mid a\geqslant 0\}$.

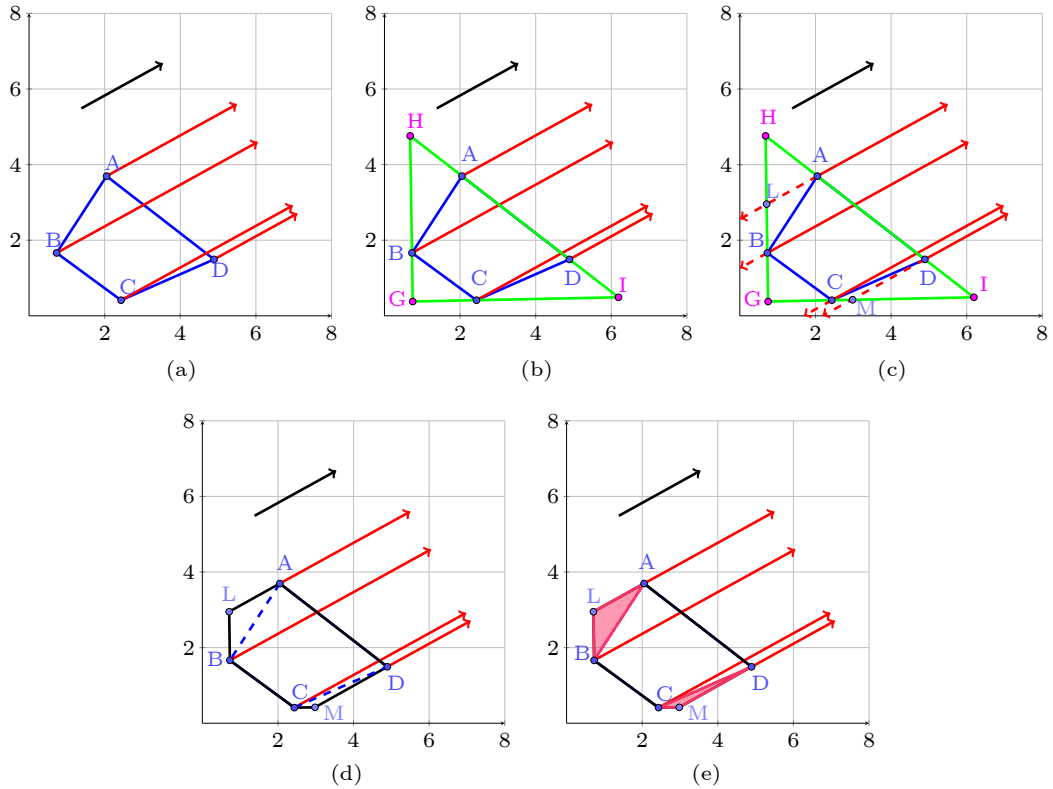


Fig.6. Illustrations for the vertex reduction algorithm of an unbounded polyhedron. The algorithm starts from the input polyhedron (a), the black arrow indicates the extreme direction), and constructs the initial simplex (b) in the first quadrant. Then preprocessing is done to ensure the infinitesimal requirement because of the boundlessness (c), (d). Lastly, the polyhedron is refined with the truncation iteration as AoYu (e). (a) Input polyhedron. (b) Initialization. (c) Preprocessing. (d) Cutting infinity. (e) Refinement.

formation, or is applied to a ReLU operation. For this reason, we can tentatively "discard" some polyhedra whose dimensions are lower than some threshold $d$. If we find that

$$\dim(f_{\mathcal{N}}^{\sharp,\,d}(\llbracket\varphi\rrbracket_{n,\,k}) \cap \llbracket\psi\rrbracket_{m,\,k}) \geqslant d$$

holds, we can be confident that such an optimization strategy is promising, where $f_{\mathcal{N}}^{\sharp,\,d}$ is the adaptation of $f_{\mathcal{N}}^{\sharp}$ with such discarding. Otherwise, we need to set a smaller threshold $d^{'}$, and a compensatory computation for some discarded parts is required, which means an economical computation only involving the polyhedra whose dimensions fall in between $d^{'}$ and $d$.

Note that such a strategy only works for quantitative model checking. Also, we in what follows do not explicitly distinguish $f_{\mathcal{N}}^{\sharp,\,d}$ and $f_{\mathcal{N}}^{\sharp}$.

Because we have introduced abstraction during the propagation, to pursue the accuracy, we need to do the refinement. Note that this is required only in the following situations.

1) For doing qualitative model checking, when $f_{\mathcal{N}}^{\sharp}(\llbracket\varphi\rrbracket)_{n,\,k} \not\subseteq \llbracket\psi\rrbracket_{m,\,k}$ holds.

2) For the quantitative case, once we find that the majority part of $f_{\mathcal{N}}^{\sharp}(\llbracket\varphi\rrbracket)_{n,\,k}$ is bounded and $\dim(f_{\mathcal{N}}^{\sharp}(\llbracket\varphi\rrbracket)_{n,\,k}) = \dim(f_{\mathcal{N}}^{\sharp}(\llbracket\varphi\rrbracket)_{n,\,k} \cap \llbracket\psi\rrbracket_{m,\,k})$ holds.

Indeed, let $\mathcal{X} = f_{\mathcal{N}}^{\sharp}(\llbracket\varphi\rrbracket)_{n,\,k}$ and $\mathcal{Y} = \mathcal{X} \cap \llbracket\psi\rrbracket_{m,\,k}$, other cases for quantitative model checking can be handled as follows.

1) For the case $\dim \mathcal{Y} < \dim \mathcal{X}$, or $\dim \mathcal{Y} = \dim \mathcal{X}$ and the majority of part of $\mathcal{Y}$ is bounded but not the case of $\mathcal{X}$, then we surely have $(\varphi\ \{\mathcal{N}\}\ \psi)_k = 0$.

2) If $\dim \mathcal{X} = \dim \mathcal{Y}$ and both the majority of parts of $\mathcal{X}$ and $\mathcal{Y}$ are unbounded, then the value $(\varphi\ \{\mathcal{N}\}\ \psi)_k$ can be computed using the symbolic approach introduced in Section 3.

Key techniques for doing refinement are the Monte Carlo sampling and the backward propagation. Let us elaborate them in the followings.

*Backward Propagation.* Just recall the construction of the rigorous propagation, for any given space $\mathcal{X}$, we may compute $f_{\mathcal{N}}^{-1}(\mathcal{X})$ in a backward manner. This is simply based on the following trivial facts.

1) First, for a mapping $g : \boldsymbol{v} \mapsto \boldsymbol{v} + \boldsymbol{b}$, where $\boldsymbol{b}$ is a constant vector, we have $g^{-1}(\boldsymbol{v}) = \boldsymbol{v} - \boldsymbol{b}$, and hence $g^{-1}(\mathcal{V}) = \{\boldsymbol{v} - \boldsymbol{b} \mid \boldsymbol{v} \in \mathcal{V}\}$.

2) Then $\boldsymbol{ReLU}_i^{-1}(\{\boldsymbol{v}\})$ is defined as:

$$\begin{cases} \{\boldsymbol{v}\}, & v_i > 0, \\ \emptyset, & v_i < 0, \\ \{(v_1, \ldots, v_{i-1}, u, v_{i+1}, \ldots, v_t)^{\mathrm{T}} \mid u \leqslant 0\}, & v_i = 0, \end{cases}$$

where $\boldsymbol{v} = (v_1, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_t)^{\mathrm{T}}$.

3) $\boldsymbol{ReLU}_X^{-1} = \boldsymbol{ReLU}_{i_r}^{-1} \circ \cdots \circ \boldsymbol{ReLU}_{i_2}^{-1} \circ \boldsymbol{ReLU}_{i_1}^{-1}$, if $X = \{i_1, i_2, \ldots, i_r\}$.

Thus, together with Theorem 1, we can obtain the preimage $f_{\mathcal{N}}^{-1}(\mathcal{X})$ in a step-wise manner. Practically, more than one preimage would be generated during this process, which may be further involved in the following backward propagation. To mitigate this case, we retain a point set (with a limitation of the total number) sampling from the set regions during the propagation procedure. If the point set becomes $\emptyset$ after one backward step, the process is terminated early (called the early stop mechanism). In what follows, we say that $\boldsymbol{p}$ is feasible if $f_{\mathcal{N}}^{-1}(\{\boldsymbol{p}\}) \cap [\![\varphi]\!]_{n,\,k} \neq \emptyset$.

*Refinement for Qualitative Model Checking.* For qualitative model checking, we need to decide if

$$f_{\mathcal{N}}^{-1}\left(f_{\mathcal{N}}^{\sharp}([\![\varphi]\!]_{n,\,k}) \setminus [\![\psi]\!]_{m,\,k}\right) \cap [\![\varphi]\!]_{n,\,k} = \emptyset$$

holds. Instead of proceeding in a monolithic fashion, we can partition the postimage into several parts (e.g., a simplex), and each time just compute the preimage of one simplex. The verification can be terminated immediately if a non-empty preimage of any part is detected.

*Refinement for Quantitative Model Checking.* To evaluate how often a given property holds, we adopt a Monte Carlo sampling based method for quantitative model checking, which is widely utilized in the field of statistical model checking[32], ranging from safety verification[33], and risk analysis[34], to resilience assessment[35]. More precisely, for an error $\epsilon$ specified by users, the estimate $\hat{p}$ does not lie outside the ground truth $\pm\epsilon$ with a confidence probability $1-\delta$. Suppose that a system has true probability $p$ of satisfying the given property, and then according to [32],

$$\mathrm{Prob}(|\hat{p} - p| \geqslant \epsilon) \leqslant \delta, \quad \text{if } N \geqslant \lceil (\ln 2 - \ln \delta)/(2\epsilon^2) \rceil,$$

where $N$ is the sample number and the required bound of $N$ is termed the Chernoff bound[36]. Conversely, a given sample number $N$ guarantees a confident probability $\delta = 2\mathrm{e}^{-2N\epsilon^2}$ with respect to error $\epsilon$.

Let $\mathcal{B}$ be a hyper-cube subsuming the majority of part of $f^{\sharp}([\![\varphi]\!]_{n,\,k})$ but with the same dimension (we can find such $\mathcal{B}$ with the approach introduced in the abstraction algorithm). We initialize two integer variables $n_\varphi$ and $n_\psi$ with 0, and we uniformly sample within the region $\mathcal{B}$. For each sample $\{\boldsymbol{p}\}$, we do the followings:

1) decide the feasibility of $\boldsymbol{p}$ using the backward propagation;

2) if $\boldsymbol{p}$ is feasible, we increase $n_\varphi$ by 1; and if in addition $p \models \psi$ holds, we also add 1 to $n_\psi$.

We finally take $n_\psi/n_\varphi$ as the corresponding value of $(\varphi\{\mathcal{N}\}\psi)_k$, once a required quantity of samples has been testified, such as the Chernoff bound with respect to a tolerant error and a confidence probability.

In summary, Fig.7 depicts the whole verification framework against RNNs, with the main techniques

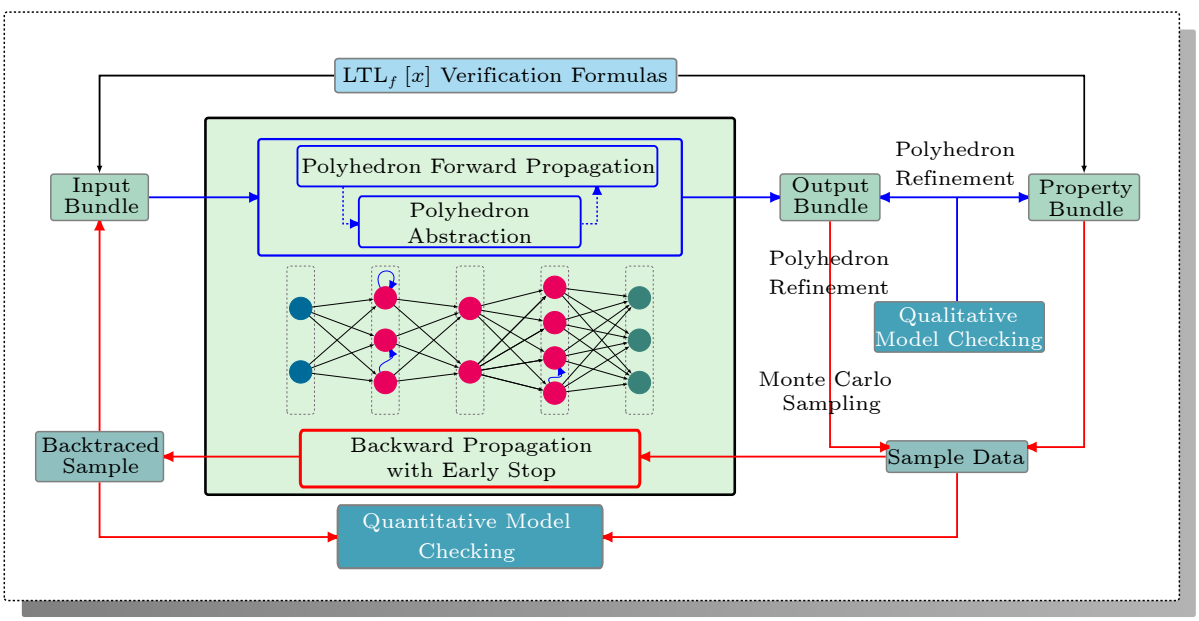


Fig.7. Overview of the verification framework.

utilized in the framework. Along with the blue arrows, the forward propagation completes and the qualitative model checking finishes, whereas, the quantitative result is obtained via the Monte Carlo sampling and the backward propagation along the red arrows.

*Complexity and Scalability Analysis.* The analysis is conducted from two aspects, i.e., the original verification framework without the polyhedron abstraction and the speculative optimization strategy, and the verification framework with the polyhedron abstraction and the speculative optimization strategy. We consider an $L$-layer RNN with layer width $d_\ell$, $\ell \in \{1, 2, \cdots, L\}$. Its input sequence length is $k$ and the input vector is in the shape $\mathbb{R}^{d_1 \times k}$. Assume that there include $p_0$ extreme vertices and $v_0$ extreme directions depicting the input region according to the pre-condition $\varphi$. The whole verification process takes $k \times (L-1)$ steps of forward polyhedron propagation totally.

Firstly, the complexity and scalability are analyzed for the original verification framework without the polyhedron abstraction and the speculative optimization. In this case, supposing that there exist $p$ extreme vertices and $v$ extreme directions representing the polyhedron, their numbers would increase quadratically at most at the end of the following one-step forward propagation, in $O(p(v+p))$ and $O(v^2)$ time, respectively. The number of extreme directions in space $\mathbb{R}^n$ is bounded by $n$ and the input space is $\mathbb{R}^{d_1 \times k}$, and then the numbers of extreme vertices and extreme directions would increase to $v_0^{2^{k(L-1)}}$ and $d_1 \times k$, respectively, after the whole verification process in the worst case, which is computationally prohibitive with respect to large input width $d_1$ and input sequence length $k$, and greatly limits the scalability of the proposed framework.

Secondly, we take the polyhedron abstraction and the speculative optimization strategy into account. The overall complexity of the polyhedron abstraction is the same as that of Yu's algorithm[30], i.e., about $O(Mn^5)$, where $n$ is the space dimension and $M$ is the iteration number. The iteration number significantly depends on the approximation precision. As for the speculative optimization, it is with complexity $O(s)$ and $s$ is the polyhedron number within a bundle. It can be seen that such abstraction is not a lightweight complexity algorithm; however, it reduces the vertex number significantly and the following forward propagation duration further. Moreover, the speculative optimization also makes great contributions to the scalability of the verification framework.

In summary, the original verification framework can achieve exact polyhedron propagation, yet with high computational complexity and limited scalability. However, the polyhedron abstraction and speculative optimization techniques alleviate the dilemma to a great extent, with a bit of precision sacrifice.

## 5 Experiments

In this section, we exhibit the experimental results of the proposed approach and place more emphasis on demonstrating the expressive capabilities of $\mathrm{LTL}_f[\boldsymbol{x}]$. The verified properties are categorized into non-temporal properties and temporal ones herein, where the former includes common properties in the existing work, such as robustness, adversarial examples, and output reachability, and the latter includes properties that have been hardly considered so far. Moreover, we also record some illustrative experiment details to highlight the quality and performance of the key techniques proposed in the verification framework.

A prototype toolkit BPMC[2] has been developed based on our verification framework. All the experiments herein are run on the platform with Windows 11 system and the 11th Gen Intel® Core™ i7-11800H @ 2.30 GHz and RAM 16 GB.

*Experiment Setting.* In our experiments, five academic RNN instances $\mathcal{N}_1$, $\mathcal{N}_2$, $\mathcal{N}_3$, $\mathcal{N}_4$, and $\mathcal{N}_5$ are constructed, which are all decision networks for sequence classification problems, to illustrate the feasibility of the proposed framework. Parameters of all the networks are randomly generated. The networks' structures and input sequence lengths $k$ are shown in Table 1. Taking the network $\mathcal{N}_3$ as an example, its input-width and output-width are 2 and 3, respectively, and it has four hidden layers with dimensions 7, 10, 10 and 7, respectively.

**Table 1.** Network Structures and Input Sequence Lengths

| Network | Network Structure | Input Length |
|---|---|---|
| $\mathcal{N}_1$ | 2-3-3-2 | 3 |
| $\mathcal{N}_2$ | 2-7-10-10-7-3 | 3 |
| $\mathcal{N}_3$ | 8-6-6-2 | 6 |
| $\mathcal{N}_4$ | 1-2-2 | 3 |
| $\mathcal{N}_5$ | 1-6-2 | 3 |

### 5.1 Verification of Non-Temporal Properties

According to the taxonomy presented in [7], the

main non-temporal properties of neural networks and their corresponding descriptions are displayed as follows.

*Robustness.* It generally means the evaluation of effects induced by perturbations affecting the inputs. More clearly, it analyzes whether the classification results remain unchanged if an $\epsilon$-perturbation is applied to the inputs.

*Adversarial Examples.* They refer to the input samples that cause neural networks to make a false prediction (or, classification).

*Output Reachability.* It indicates whether an output range or a state can be reached, which is often used to verify the safety of neural networks.

In this subsection, we study these properties against the recurrent neural networks $\mathcal{N}_1$, $\mathcal{N}_2$ and $\mathcal{N}_3$.

*Robustness.* As for robustness, we randomly generate the input sequences $\boldsymbol{\rho}$ for the networks $\mathcal{N}_1$, $\mathcal{N}_2$ and $\mathcal{N}_3$, as shown in Table 2, which are classified into the 1st, the 3rd, and the 1st class via the corresponding network's prediction, respectively.

**Table 2.** Network Input Sequences

| Network | Original Input Sequence |
|---|---|
| $\mathcal{N}_1$ | $(-0.36, 0.5)^{\mathrm{T}}$, $(-0.71, 1.38)^{\mathrm{T}}$, $(1.79, -2.33)^{\mathrm{T}}$ |
| $\mathcal{N}_2$ | $(-1.24, 0.48)^{\mathrm{T}}$, $(0.61, 0.92)^{\mathrm{T}}$, $(-1.7, -0.23)^{\mathrm{T}}$ |
| $\mathcal{N}_3$ | $(-3.4, 8.0, -5.0, 9.5, 3.6, -1.4, -0.4, -0.7)^{\mathrm{T}}$, $(-4.1, -0.2, 9.6, -3.3, -3.7, -5, -11.2, 2.8)^{\mathrm{T}}$, $(6.7, 11.6, -2.8, 6.8, 0.2, 7.6, 7.1, 5.1)^{\mathrm{T}}$, $(-7.5, 2.2, -4.8, -3.3, 1.8, -0.6, 5.9, 7.2)^{\mathrm{T}}$, $(-3.6, 8.8, 1.8, -5.6, -0.4, 0.9, -0.1, 8.7)^{\mathrm{T}}$, $(-5.2, -0.6, -3.9, 3.9, 5.4, 2.5, -1.9, -0.9)^{\mathrm{T}}$. |

Qualitative (quali. res.) and quantitative (quan. res.) model checking results on those networks are given in Table 3. The verification duration on network $\mathcal{N}_1$, $\mathcal{N}_2$ and $\mathcal{N}_3$ averagely takes 1 second, 6 seconds, and 15 seconds, respectively. Taking the fifth row as an example, it declares that *"for input sequence* $\boldsymbol{\rho}$ *on network* $\mathcal{N}_1$*, adding* 0.05 *to the* 1*st element of the input on the* 3*rd timestep* $((i, j)/i{=}(3, 1))$ *is robust* (100%)*, while subtracting* 0.1 *from both elements of the input on the* 3*rd timestep* $((i, j/)i{=}3)$ *is not robust, with* 99.3% *of the outputs satisfying the robustness."*

The pre-condition and post-condition corresponding to the 2nd–5th (resp. rightmost four) columns can be formulated as

$$\boldsymbol{e}_j^{\mathrm{T}}(\mathsf{X}^i\boldsymbol{x} - \boldsymbol{M}_i \cdot \mathrm{Jux}(\boldsymbol{\rho})) < \epsilon$$

$$(\text{resp. } \mathsf{X}^i\boldsymbol{x} - \boldsymbol{M}_i \cdot \mathrm{Jux}(\boldsymbol{\rho}) < (\epsilon, \epsilon)^{\mathrm{T}}),$$

respectively and $\mathsf{X}^5\boldsymbol{x} > \mathsf{X}^6\boldsymbol{x}$, where $\boldsymbol{M}_i$ is the matrix $\left(\underbrace{\boldsymbol{0}\ \cdots\ \boldsymbol{0}}_{i-1}\ \boldsymbol{I}_2\ \underbrace{\boldsymbol{0}\ \cdots\ \boldsymbol{0}}_{K-i}\right)$ and the vectors $\boldsymbol{e}_1 = (1, 0)^{\mathrm{T}}$, $\boldsymbol{e}_2 = (0, 1)^{\mathrm{T}}$.

*Adversarial Examples.* For an unrobust input region, we now intend to detect the existence of adversarial example(s) for a given perturbation $\epsilon$. The verification results are shown in Table 3. Herein, we focus on the network $\mathcal{N}_1$ and the others are similar. In this experiment, the pre-condition is

$$-\epsilon < \bigwedge_{i=1}^{K} \bigwedge_{j=1,2} \boldsymbol{e}_j^{\mathrm{T}}(\mathsf{X}^i\boldsymbol{x} - \boldsymbol{M}_i \cdot \mathrm{Jux}(\boldsymbol{\rho})) < \epsilon,$$

and the post-condition is $\mathsf{X}^5\boldsymbol{x} > \mathsf{X}^6\boldsymbol{x}$. Experimental results are shown in Table 4. For a concrete $\epsilon$ value (the 3rd column), it yields the concrete adversarial examples, and elements that have been changed (in comparison with $\boldsymbol{\rho}$) are written in bold (in the 2nd column), and the number of changed input elements is in the 1st column.

*Reachability.* In this setting, we say that an input is unsafe if it yields an output that is "close" enough to both categories, namely, specified with the post-condition

$$-\epsilon < ((\boldsymbol{e}_1^{\mathrm{T}} - \boldsymbol{e}_2^{\mathrm{T}})(\mathsf{X}^3\boldsymbol{x})) < \epsilon,$$

**Table 3.** Verification Results on the Robustness Property

| Network | $(i, j)/i$ | $\epsilon$ | Quali. Res. | Quan. Res. | $(i, j)/i$ | $\epsilon$ | Quali. Res. | Quan. Res. |
|---|---|---|---|---|---|---|---|---|
| $\mathcal{N}_1$ | (3, 1) | –0.05 | Sat | 1.000 | 1 | +0.05 | Sat | 1.000 |
| | (3, 1) | –0.10 | Sat | 1.000 | 2 | +0.10 | Sat | 1.000 |
| | (3, 1) | –0.20 | Sat | 1.000 | 2 | –0.45 | Unsat | 0.996 |
| | (3, 1) | +0.05 | Sat | 1.000 | 3 | –0.10 | Unsat | 0.993 |
| | (3, 1) | +0.10 | Unsat | 0.705 | 3 | +0.10 | Unsat | 0.870 |
| $\mathcal{N}_2$ | (1, 1) | +0.10 | Unsat | 0.884 | 1 | –0.01 | Sat | 1.000 |
| | (1, 2) | –0.20 | Unsat | 0.717 | 1 | –0.05 | Sat | 1.000 |
| | (1, 2) | –0.15 | Unsat | 0.725 | 2 | –0.10 | Sat | 1.000 |
| $\mathcal{N}_3$ | 3 | –0.10 | Sat | 1.000 | 5 | –0.10 | Sat | 1.000 |
| | 4 | –0.10 | Sat | 1.000 | 6 | –0.10 | Sat | 1.000 |

**Table 4.** Model Checking Results on Adversarial Examples

| Changed Unit (s) | Adversarial Example | $\epsilon$ |
|---|---|---|
| 1 | $(-0.36, 0.50)^{\mathrm{T}}$, $(-0.71, 1.38)^{\mathrm{T}}$, $(1.79, \mathbf{-2.39})^{\mathrm{T}}$ | 0.06 |
| 1 | $(-0.36, \mathbf{0.35})^{\mathrm{T}}$, $(-0.71, 1.38)^{\mathrm{T}}$, $(1.79, -2.33)^{\mathrm{T}}$ | 0.15 |
| 2 | $(-0.36, \mathbf{0.47})^{\mathrm{T}}$, $(-0.71, 1.38)^{\mathrm{T}}$, $(1.79, \mathbf{-2.39})^{\mathrm{T}}$ | 0.09 |
| 2 | $(-0.36, 0.50)^{\mathrm{T}}$, $(-0.71, \mathbf{1.47})^{\mathrm{T}}$, $(\mathbf{1.87}, -2.33)^{\mathrm{T}}$ | 0.17 |
| 3 | $(-0.36, 0.50)^{\mathrm{T}}$, $(-0.71, \mathbf{1.43})^{\mathrm{T}}$, $(\mathbf{1.85}, \mathbf{-2.27})^{\mathrm{T}}$ | 0.15 |
| 3 | $(-0.36, \mathbf{0.44})^{\mathrm{T}}$, $(-0.71, 1.38)^{\mathrm{T}}$, $(\mathbf{1.73}, \mathbf{-2.38})^{\mathrm{T}}$ | 0.17 |

but with the pre-condition True with respect to network $\mathcal{N}_1$. Taking $\epsilon = 0.02$, we detect an unsafe input $\boldsymbol{\rho}' = (-0.36, 0.50)^{\mathrm{T}}, (-0.71, 1.38)^{\mathrm{T}}, (1.79, -2.43)^{\mathrm{T}}$ for $\mathcal{N}_1$.

### 5.2 Verification of Temporal Properties

Compared with other networks, the exclusive structural feature of RNNs lies in the memory units, which is the reason for their utilization in sequence processing. Therefore, the temporal properties upon RNNs can be defined, whereas they have been paid little attention so far. We conduct experiments by verifying some temporal properties indicating convergence to a great extent. In this subsection, the experiments are carried out on networks $\mathcal{N}_4$ and $\mathcal{N}_5$. The first property is specified in the input sequence, declaring that "*whenever the input is monotonically increasing, would the sample be divided into the second class*?" The second property is to justify "*if the input at the second time step is at most* 0*, would the probability that the sample belongs to the second class decrease monotonically*?" and the last one is to verify *"whether the sample would be classified into the second class if the input at the first time step is smaller than the one at the second time step*?"

The first property and the second property are verified on network $\mathcal{N}_4$ and the last one is on $\mathcal{N}_5$. As illustrated in Table 5, the first property is Unsat with 0.174 of the outputs satisfying the post-condition, whereas the second is Unsat with the probability 0.694 3. As for the last property, it completely holds on network $\mathcal{N}_5$. The verification duration for these properties takes about 0.04, 0.15, and 0.4 seconds respectively.

### 5.3 Performance Demonstrations

In this subsection, except the above-mentioned expressive capabilities of the specification logic $\mathrm{LTL}_f[\boldsymbol{x}]$ and the efficacy of our proposed verification framework, we show some running details of the verification processes to demonstrate the quality and effectiveness of the key technique, i.e., the polyhedra abstraction with speculative optimization.

Herein, we make some comparisons between the original verification process and the one coupled with the polyhedron abstraction and the speculative optimization in terms of computation time and memory, with respect to the verification examples on network $\mathcal{N}_3$ in Table 3. The computation time and the total memory space of the original and integrated cases are listed in Table 6, corresponding to "-o" and "-a" labeled columns, respectively.

On the one hand, it can be observed that with the polyhedron abstraction and the speculative optimization techniques, the verification process can be accelerated greatly, which mainly stems from the fact that the polyhedron abstraction reduces the following computational burden and the speculative optimization discards lower-dimensional polyhedra during the verification process.

On the other hand, the verification process, combined with polyhedron abstraction and speculative optimization, takes a little more memory space compared with the original verification framework. It results from the fact that the polyhedron abstraction process occupies more memory, while the speculative optimization has discarded some polyhedra in the following process.

Considering the performance of the verification duration and memory spaces, polyhedron abstraction and speculative optimization are necessary to be adopted for verification acceleration. Moreover, as the final verification conclusions show, it is notable that robustness still holds in all the cases even when some over-approximation is introduced during the abstraction process. However, it is of great importance to carefully balance the verification precision and computation efficiency of the polyhedron abstraction pro-

**Table 5.** Experimental Results of Model Checking upon Temporal Properties

| Property | Pre-Condition | Post-Condition | Proportion |
|---|---|---|---|
| Property 1 | $\forall i.1 \leqslant i \leqslant 2 \rightarrow \mathsf{X}^{i+1}\boldsymbol{x} \geqslant \mathsf{X}^{i}\boldsymbol{x}$ | $\boldsymbol{e}_1^{\mathrm{T}}(\mathsf{X}^3\boldsymbol{x}) \leqslant \boldsymbol{e}_2^{\mathrm{T}}(\mathsf{X}^3\boldsymbol{x})$ | 0.174 0 |
| Property 2 | $\mathsf{X}^2\boldsymbol{x} \leqslant 0$ | $\forall i.1 \leqslant i \leqslant 2 \rightarrow (\boldsymbol{e}_1^{\mathrm{T}}(\mathsf{X}^{i+1}\boldsymbol{x}) < \boldsymbol{e}_1^{\mathrm{T}}(\mathsf{X}^{i}\boldsymbol{x}))$ | 0.694 3 |
| Property 3 | $\mathsf{X}^1\boldsymbol{x} \leqslant \mathsf{X}^2\boldsymbol{x}$ | $\boldsymbol{e}_1^{\mathrm{T}}(\mathsf{X}^3\boldsymbol{x}) \leqslant \boldsymbol{e}_2^{\mathrm{T}}(\mathsf{X}^3\boldsymbol{x})$ | 1.000 0 |

**Table 6.** Comparisons on Computation Time and Memory

| $i$ | $\text{Time}_o$ (s) | $\text{Time}_a$ (s) | $\text{Mem}_o$ (MB) | $\text{Mem}_a$ (MB) |
|---|---|---|---|---|
| 3 | 15.405 | 1.181 | 93.2 | 95.2 |
| 4 | 13.974 | 1.155 | 93.6 | 95.2 |
| 5 | 14.500 | 1.112 | 93.3 | 95.0 |
| 6 | 13.701 | 1.115 | 93.9 | 95.4 |

cess according to practical cases.

## 6 Related Work

So far, the work on verification of RNNs is rather scarce. The most common approach is to convert RNNs to FNN-similar networks and verify them with the mature methods for the latter. Akintunde *et al.*[15] first proposed the idea of unrolling, which concatenates the structure of an RNN for different timesteps, thus degenerating an RNN to an FNN-like network. To overcome the difficulty caused by scale explosion, Jacoby *et al.*[16] used invariant inference to deal with the loop structure and considered the loop as an input element added to the input layer. In this paper, we utilize the "unrolling" idea and leverage a temporal logic specification to reason about the properties of RNN.

Different from converting to FNN-like networks, Ko *et al.*[17] proposed the POPQORN algorithm based on linear approximation, and as far as we know, it is the first framework to provide a quantified robustness evaluation of RNNs. Later, Du *et al.*[18] presented Cert-RNN based on the zonotope abstraction, and Ryou *et al.*[19], very recently, proposed Prover relying on linear programming and polyhedron abstraction, both of which obtain more precise and scalable results than the prior work. The listed work mainly focuses on robustness, and ignores the temporal properties of the input/output sequences of RNNs. In our work, we take the entire input/output sequences into consideration and verify some non-trivial temporal properties as well. Moreover, the quantified robustness evaluations specify the robust input regions but to what extent the robustness property holds in unrobust input ones is not taken into account, both of which are tackled in our framework. In this paper, the motivation that we select polyhedra as abstract domains for the RNN verification, instead of zonotopes, polytopes or star sets, is mainly three-fold. Firstly, polyhedra are compatible with our specification language, which means that the conversion between them is easy to implement and understand. Secondly, polyhedron sets (i.e., polyhedron bundles defined in the paper) are closed under the ReLU operator. While zonotopes do not hold this property, meaning that their computation introduces more wrapping effect in the verification. Last but not least, polyhedra can depict unbounded constrained regions, but polytopes and star sets cannot. Besides, a bounded polyhedron is a polytope essentially and any bounded polyhedra can be represented as star sets[37].

The RNN verification scheme proposed by Zhang *et al.* in 2020[20], based on reachability analysis, is more related to our paper. RNNs are abstractly interpreted by polytope propagation and fixed point analysis, and then RNNs are verified against cognitive tasks. In this paper, we circumvent the exponential increase in the vertex number through the whole polyhedron propagation with V-representation and verify the properties quantitatively.

Statistic model checking on automata in [22], another related work to ours, provides probabilistic results of the robustness verification of RNNs. However, the results of [22] based on sampling are approximations of the ground truth (cannot specify whether an input region is robust or not). Our proposed framework can derive qualitative results to show the property satisfiability, not only limited to robustness, and provide probabilistic (quantitative) results when properties do not hold.

In addition, researchers also attempt to utilize automata and other computational models to conduct formal modeling of RNNs. RNN behaviors can also be modeled and verified by labeled transition systems, deterministic finite automata and rule extraction, and context-free grammars and probabilistic automata. The early automata extraction technology mainly used hierarchical clustering analysis to analyze the continuous state space of recurrent neural networks[38], and some scholars proposed a sampling-based method to extract automata[39].

## 7 Conclusions

In this paper, we proposed a specification language $\text{LTL}_f[\boldsymbol{x}]$ and an alternative verification framework unifying qualitative and quantitative model checking for RNNs. The key data structure utilized to do the model checking is the polyhedron abstract domain and the polyhedron forward propagation is introduced for ReLU RNNs. To alleviate the vertex explosion during the propagation process, we put forward the (dimension-preserving) polyhedron abstrac-

tion and leveraged polyhedron refinement to achieve most precise results possible for both qualitative and quantitative verification, based on the Monte Carlo sampling. A prototype tool named BPMC$^2$ was implemented to examine the feasibility of the proposed algorithms, taking both non-temporal and temporal properties into account. The tool verified the given properties successfully with considerable memory space and computation time.

Based on the polyhedron abstract domain and $\mathrm{LTL}_f[\boldsymbol{x}]$, our verification framework theoretically resolves the technical intractability of ReLU RNN validation. However, further optimization on the scalability is still needed for its deployment and application in practical scale networks, such as more efficient data structures and organization, more reasonable approximation strategy balancing precision and efficiency, and systematic integration of floating-point computation. Extending our verification framework on other activation functions and RNN variants and verifying the properties with unbounded input lengths are also promising research directions in the future.

**Conflict of Interest** The authors declare that they have no conflict of interest.

## References


[1] Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z H, Karpathy A, Khosla A, Bernstein M, Berg A C, Fei-Fei L. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 2015, 115(3): 211–252. DOI: 10.1007/s11263-015-0816-y.

[2] Pennington J, Socher R, Manning C D. GloVe: Global vectors for word representation. In *Proc. the 2014 Conference on Empirical Methods in Natural Language Processing*, Oct. 2014, pp.1532–1543. DOI: 10.3115/v1/d14-1162.

[3] Hinton G, Deng L, Yu D, Dahl G E, Mohamed A R, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Sainath T N, Kingsbury B. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 2012, 29(6): 82–97. DOI: 10.1109/MSP.2012.2205597.

[4] Liu X W, Zhu X Z, Li M M, Wang L, Tang C, Yin J P, Shen D G, Wang H M, Gao W. Late fusion incomplete multi-view clustering. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2019, 41(10): 2410–2423. DOI: 10.1109/TPAMI.2018.2879108.

[5] Urmson C, Whittaker W. Self-driving cars and the urban challenge. *IEEE Intelligent Systems*, 2008, 23(2): 66–68. DOI: 10.1109/mis.2008.34.

[6] Litjens G, Kooi T, Bejnordi B E, Setio A A A, Ciompi F, Ghafoorian M, van der Laak J A W M, van Ginneken B, Sánchez C I. A survey on deep learning in medical image analysis. *Medical Image Analysis*, 2017, 42: 60–88. DOI: 10.1016/j.media.2017.07.005.

[7] Huang X W, Kroening D, Ruan W J, Sharp J, Sun Y C, Thamo E, Wu M, Yi X P. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 2020, 37: 100270. DOI: 10.1016/j.cosrev.2020.100270.

[8] Molnar C, Casalicchio G, Bischl B. Interpretable machine learning—A brief history, state-of-the-art and challenges. In *Proc. the 2020 Workshops of the European Conference on Machine Learning and Knowledge Discovery in Databases*, Sept. 2020, pp.417–431. DOI: 10.1007/978-3-030-65965-3_28.

[9] Goodfellow I J, Shlens J, Szegedy C. Explaining and harnessing adversarial examples. In *Proc. the 3rd International Conference on Learning Representations*, May 2015.

[10] Papernot N, McDaniel P, Jha S, Fredrikson M, Celik Z B, Swami A. The limitations of deep learning in adversarial settings. In *Proc. the 2016 IEEE European Symposium on Security and Privacy*, Mar. 2016, pp.372–387. DOI: 10.1109/EuroSP.2016.36.

[11] Katz G, Barrett C W, Dill D L, Julian K, Kochenderfer M J. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Proc. the 29th International Conference on Computer Aided Verification*, Jul. 2017, pp.97–117. DOI: 10.1007/978-3-319-63387-9_5.

[12] Gehr T, Mirman M, Drachsler-Cohen D, Tsankov P, Chaudhuri S, Vechev M. AI2: Safety and robustness certification of neural networks with abstract interpretation. In *Proc. the 2018 IEEE Symposium on Security and Privacy*, May 2018, pp.3–18. DOI: 10.1109/sp.2018.00058.

[13] Singh G, Gehr T, Püschel M, Vechev M. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 2019, 3(POPL): 41. DOI: 10.1145/3290354.

[14] Liu W W, Song F, Zhang T H R, Wang J. Verifying ReLU neural networks from a model checking perspective. *Journal of Computer Science and Technology*, 2020, 35(6): 1365–1381. DOI: 10.1007/s11390-020-0546-7.

[15] Akintunde M E, Kevorchian A, Lomuscio A, Pirovano E. Verification of RNN-based neural agent-environment systems. In *Proc. the 33rd AAAI Conference on Artificial Intelligence*, Jan. 27– Feb. 1, 2019, pp.6006–6013. DOI: 10.1609/aaai.v33i01.33016006.

[16] Jacoby Y, Barrett C, Katz G. Verifying recurrent neural networks using invariant inference. In *Proc. the 18th International Symposium on Automated Technology for Verification and Analysis*, Oct. 2020, pp.57–74. DOI: 10.1007/978-3-030-59152-6_3.

[17] Ko C Y, Lyu Z Y, Weng L, Daniel L, Wong N, Lin D H. POPQORN: Quantifying robustness of recurrent neural networks. In *Proc. the 36th International Conference on Machine Learning*, Jun. 2019, pp.3468–3477.

[18] Du T Y, Ji S L, Shen L J, Zhang Y, Li J F, Shi J, Fang C F, Yin J W, Beyah R, Wang T. Cert-RNN: Towards certifying the robustness of recurrent neural networks. In *Proc. the 2021 ACM SIGSAC Conference on Computer*

*and Communications Security*, Nov. 2021, pp.516–534. DOI: 10.1145/3460120.3484538.

[19] Ryou W, Chen J Y, Balunovic M, Singh G, Dan A, Vechev M. Scalable polyhedral verification of recurrent neural networks. In *Proc. the 33rd International Conference on Computer Aided Verification*, Jul. 2021, pp.225–248. DOI: 10.1007/978-3-030-81685-8_10.

[20] Zhang H C, Shinn M, Gupta A, Gurfinkel A, Le N, Narodytska N. Verification of recurrent neural networks for cognitive tasks via reachability analysis. In *Proc. the 24th European Conference on Artificial Intelligence*, Aug. 29–Sept. 8, 2020, pp.1690–1697. DOI: 10.3233/FAIA200281.

[21] Vengertsev D, Sherman E. Recurrent neural network properties and their verification with Monte Carlo techniques. In *Proc. the 34th AAAI Conference on Artificial Intelligence*, Feb. 2020, pp.178–185.

[22] Khmelnitsky I, Neider D, Roy R, Xie X, Barbot B, Bollig B, Finkel A, Haddad S, Leucker M, Ye L N. Property-directed verification and robustness certification of recurrent neural networks. In *Proc. the 19th International Symposium on Automated Technology for Verification and Analysis*, Oct. 2021, pp.364–380. DOI: 10.1007/978-3-030-88885-5_24.

[23] Kalra N, Paddock S M. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice*, 2016, 94: 182-193. DOI: 10.1016/j.tra.2016.09.010.

[24] Dahnert M, Hou J, Nießner M, Dai A. Panoptic 3D scene reconstruction from a single RGB image. In *Proc. the 35th International Conference on Neural Information Processing Systems*, Dec. 2021, Article No. 633.

[25] Wang J X, Wang K C, Rudzicz F, Brudno M. Grad2Task: Improved few-shot text classification using gradients for task representation. In *Proc. the 35th International Conference on Neural Information Processing Systems*, Dec. 2021, Article No. 501.

[26] Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators. *Neural Networks*, 1989, 2(5): 359–366. DOI: 10.1016/0893-6080(89)90020-8.

[27] Ziegler G M. Lectures on Polytopes. Springer, 1995. DOI: 10.1007/978-1-4613-8431-1.

[28] Preparata F P, Shamos M I. Computational Geometry: An Introduction. Springer, 1985. DOI: 10.1007/978-1-4612-1098-6.

[29] Bredon G E. Topology and Geometry. Springer, 1993. DOI: 10.1007/978-1-4757-6848-0.

[30] Zheng Y. Computing bounding polytopes of a compact set and related problems in *n*-dimensional space. *Computer-Aided Design*, 2019, 109: 22–32. DOI: 10.1016/j.cad.2018.12.002.

[31] Barber C B, Dobkin D P, Huhdanpaa H. The quickhull algorithm for convex hulls. *ACM Trans. Mathematical Software*, 1996, 22(4): 469–483. DOI: 10.1145/235815.235821.

[32] Legay A, Lukina A, Traonouez L M, Yang J X, Smolka S A, Grosu R. Statistical model checking. In *Computing and Software Science: State of the Art and Perspectives*, Steffen B, Woeginger G (eds.), Springer, 2019, pp.478–504. DOI: 10.1007/978-3-319-91908-9_23.

[33] Mancini T, Mari F, Melatti I, Salvo I, Tronci E, Gruber J K, Hayes B, Prodanovic M, Elmegaard L. Parallel statistical model checking for safety verification in smart grids. In *Proc. the 2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, Oct. 2018. DOI: 10.1109/smartgridcomm.2018.8587416.

[34] Wali K I, Othman S A. Schedule risk analysis using Monte Carlo simulation for residential projects. *Zanco Journal of Pure and Applied Sciences*, 2019, 31(5): 90–103. DOI: 10.21271/zjpas.31.5.11.

[35] Younesi A, Shayeghi H, Safari A, Siano P. Assessing the resilience of multi microgrid based widespread power systems against natural disasters using Monte Carlo Simulation. *Energy*, 2020, 207: 118220. DOI: 10.1016/j.energy.2020.118220.

[36] Okamoto M. Some inequalities relating to the partial sum of binomial probabilities. *Annals of the Institute of Statistical Mathematics*, 1959, 10(1): 29–35. DOI: 10.1007/bf02883985.

[37] Tran H D, Manzanas Lopez D, Musau P, Yang X D, Nguyen L V, Xiang W M, Johnson T T. Star-based reachability analysis of deep neural networks. In *Proc. the 3rd World Congress on Formal Methods*, Oct. 2019, pp.670–686. DOI: 10.1007/978-3-030-30942-8_39.

[38] Servan-Schreiber D, Cleeremans A, McClelland J L. Graded state machines: The representation of temporal contingencies in simple recurrent networks. *Machine Learning*, 1991, 7(2/3): 161–193. DOI: 10.1007/BF00114843.

[39] Schellhammer I, Diederich J, Towsey M, Brugman C. Knowledge extraction and recurrent neural networks: An analysis of an Elman network trained on a natural language learning task. In *Proc. the 1998 New Methods in Language Processing and Computational Natural Language Learning*, Jan. 1998, pp.73–78. DOI: 10.5555/1603899.1603912.

**Zhen Liang** received his B.S. degree in computer science and technology from National University of Defense Technology, Changsha, in 2019. He is currently a Ph.D. candidate at National University of Defense Technology, Changsha. His research interests include model checking, interpretation and formal verification of artificial intelligence.

**Wan-Wei Liu** received his Ph.D degree in computer science from National University of Defense Technology, Changsha, in 2009. He is a professor at National University of Defense Technology, Changsha. His research interests include theoretical computer science (particularly in automata theory and temporal logic), formal methods (particularly in verification), and software engineering.

**Fu Song** received his Ph.D. degree in computer science from University Paris-Diderot, Paris, in 2013. He is an associate professor with ShanghaiTech University, Shanghai. His research interests include formal methods and computer/AI security.

**Bai Xue** received his Ph.D. degree in applied mathematics from Beihang University, Beijing, in 2014. He is currently a research professor with the Institute of Software, Chinese Academy of Sciences, Beijing. His research interests involve formal verification of hybrid systems and AI.

**Wen-Jing Yang** received her Ph.D. degree in multi-scale modeling from Manchester University, Manchester, in 2014. She is currently an associate research fellow at the State Key Laboratory of High Performance Computing, National University of Defense Technology, Changsha. Her research interests include machine learning, robotics software, and high-performance computing.

**Ji Wang** received his PhD degree in computer science from National University of Defense Technology, Changsha, in 1995. He is currently a full professor at National University of Defense Technology, Changsha, and he is a fellow of CCF. His research interests include software engineering and formal methods.

**Zheng-Bin Pang** received his B.S., M.S., and Ph.D. degrees in computer science from National University of Defense Technology, Changsha. Currently, he is a professor at National University of Defense Technology, Changsha. His research interests range across high-speed interconnect, heterogeneous computing, and high performance computer systems.