Quantitative Verification for Neural Networks using ProbStars

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

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

Hideki Okamoto
hideki.okamoto@toyota.com
Toyota NA R&D
Ann Arbor, Michigan, USA

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

Georgios Fainekos
georgios.fainekos@toyota.com
Toyota NA R&D
Ann Arbor, Michigan, USA

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

ABSTRACT

Most deep neural network (DNN) verification research focuses on *qualitative verification*, which answers whether or not a DNN violates a safety/robustness property. This paper proposes an approach to convert qualitative verification into quantitative verification for neural networks. The resulting quantitative verification method not only can answer YES or NO questions but also can compute the probability of a property being violated. To do that, we introduce the concept of a probabilistic star (or shortly ProbStar), a new variant of the well-known star set, in which the predicate variables belong to a Gaussian distribution and propose an approach to compute the probability of a probabilistic star in high-dimensional space. Unlike existing works dealing with constrained input sets, our work considers the input set as a truncated multivariate normal (Gaussian) distribution, i.e., besides the constraints on the input variables, the input set has a probability of the constraints being satisfied. The input distribution is represented as a probabilistic star set and is propagated through a network to construct the output reachable set containing multiple ProbStars, which are used to verify the safety or robustness properties of the network. In case of a property is violated, the violation probability can be computed precisely by an exact verification algorithm or approximately by an overapproximate verification algorithm. The proposed approach is implemented in a tool named *StarV* and is evaluated using the well-known ACASXu networks and a rocket landing benchmark.

CCS CONCEPTS

• **Software and its engineering** → **Software verification**.

ACM Reference Format:
Hoang-Dung Tran, Sungwoo Choi, Hideki Okamoto, Bardh Hoxha, Georgios Fainekos, and Danil Prokhorov. 2023. Quantitative Verification for Neural Networks using ProbStars. In *HSCC '23: ACM International Conference on Hybrid Systems: Computation and Control, May 9–12, 2023, San Antonio, TX*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

HSCC '23, May 9–12, 2023, San Antonio, TX

ACM ISBN 978-1-4503-9999-9/18/06.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Formal verification of deep neural networks (DNNs) is crucial for assuring the safety and correctness of learning-enabled safety-critical autonomous systems [25]. A large body of research focused on *qualitative verification* of safety and robustness properties of DNNs under bounded input space [20, 32]. In this context, sound and complete verification algorithms [3, 16, 21, 30, 34, 36] return SAT (a property is satisfied) or UNSAT (property is not satisfied) while sound verification algorithms [27, 39] return UNSAT or UNKNOWN (due to overapproximation error) results. Although formal verification for DNNs has been an explosive research direction with many efficient tools and methods proposed recently, *quantitative verification* for DNNs focusing on computing/estimating the probability of a property being violated is still a challenging and open problem. Compared to qualitative approaches, quantitative verification provides a probabilistic guarantee for DNNs. This is very useful in practice, especially for learning-based control systems, where sensing inputs applied to a DNN controller always contain inevitable noises that can be modeled as a multivariate Gaussian (normal) distribution. Unfortunately, there are only a few quantitative verification methods proposed until now. Importantly, to the best of our knowledge, most quantitative verification methods focuses on *binary neural networks* with *quantized finite discrete inputs space* [5, 26, 40]. There is only one method proposed for the popular ReLU DNNs [10] with *continuous input space*.

In this paper, we propose an approach to convert the recent star reachability-based qualitative verification [30] into a quantitative verification approach for ReLU networks used in learning-based control systems, e.g., the supervisory control ACASXU networks [22], using probabilistic star reachability. Although the research goal is the same as [10], i.e., computing the probability of violating a safety specification, our approach is different. In [10], the authors consider ellipsoidal input space with Gaussian random variables. By propagating a confidence ellipsoid of the input through the network, the authors construct a confidence ellipsoid for the output by abstracting the nonlinear ReLU activation functions using affine and quadratic constraints. The safety of the original network can be proved by analyzing the abstracted network using semidefinite programming. In our work, we consider a more general input space with linear constraints on Gaussian random variables, i.e., a truncated Gaussian distribution. Notably, we show that the probability of a violating safety specification can be computed precisely given a truncated Gaussian distribution input instead of approximately like [10].

Our approach is built on a new set representation named probabilistic star (or shortly ProbStar), a variant of the well-known star set used in DNNs [3, 29–31, 33], and linear dynamical and hybrid systems verification [1, 2, 4, 8]. Mathematically, a ProbStar is an affine mapping of a truncated multivariate Gaussian distribution. We model the inputs as a ProbStar and propagate it through the network to construct the reachable output set containing multiple ProbStars. The reachable output set is then used to verify a user-defined safety property, and the probability of violating the safety property will be derived. In this work, two verification strategies are proposed. The first one is the exact verification, where the precise probability of safety violation is computed. The second one overapproximates the probability of safety violation by filtering out reachable intermediate sets (in the layers) with probabilities lower than a user-predefined threshold. While the exact verification algorithm rigorously explores all paths in reachability analysis, the overapproximate verification algorithm focuses on exploring paths with large probabilities. Therefore, it reduces the number of reachable sets involved and memory consumption in verification. However, the overapproximate verification algorithm must compute the probability of intermediate ProbStars before filtering, which may be costly. Our quantitative verification approach is implemented in a tool named StarV using Python. We evaluate the proposed approach on the well-known ACAS Xu networks [22] and on a neural network controller for a rocket lander benchmark for SpacEx Falcon 9 [37] trained using reinforcement learning. The experiments show that our approach successfully verifies all unsafe ACASXu networks and the rocket networks and provides the lower bound and upper bound of the networks' probability of violation.

In sum, the main contributions of this paper are:

- A new set representation named probabilistic star that is suitable for quantitative reasoning about ReLU networks.
- A new quantitative approach with exact and overapproximate verification algorithms for ReLU DNNs used in learning-based control systems using probabilistic star reachability.
- An implementation of the proposed approach in the tool named StarV that is available online for formal method and control communities to use and compare.
- A thorough evaluation and comparison of the proposed approach on a set of well-known benchmarks.

2 PRELIMINARIES

2.1 ReLU networks

A feed-forward neural network $\mathcal{F}$ may consist of one or multiple ReLU layers. Each layer ℓ, $1 \leq \ell \leq k$, consists n_ℓ neurons that are interconnected to $n_{\ell-1}$ neurons in a preceding layer. A ReLU layer performs two operations: affine mapping and ReLU activating. The affine mapping operation is a function performing a linear transformation on the output of the previous layer. The ReLU activating operation applies the ReLU activation function to the output from the affine mapping operation, $ReLU(y) = max(0, y)$. The output vector y of the layer L_ℓ is expressed as:

$$y_\ell = L_\ell(y_{\ell-1}) = ReLU(W_\ell \times y_{\ell-1} + b_\ell).$$

where $W_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ and $b_\ell \in \mathbb{R}^{n_\ell}$ are the weight matrix and the bias vector of a ℓ-th layer, respectively. Given an input vector x, the output of the ReLU network $\mathcal{F}$ is computed forwardly layer-by-layer $y = \mathcal{F}(x) = L_k((L_{k-1}, \ldots, (L_1(x))))$.

In the following, we formally define the probabilistic reachability and quantitative verification of ReLU networks. We denote $\mathcal{N}(\mu, \Sigma)$ as a multivariate Gaussian distribution with mean $\mu \in \mathbb{R}^n$ and variance $\Sigma \in \mathbb{R}^{n \times n}$ throughout the paper.

2.2 Problem formulation

In this paper, we are interested in two problems, including probabilistic reachability and quantitative verification of ReLU networks.

Definition 2.1 (Probabilistic Reachability). *Given a ReLU network $\mathcal{F}$ and a constrained probabilistic input set, $X = \{x \in \mathbb{R}^n | \, Cx \leq d \ \wedge \ x \sim \mathcal{N}(\mu, \Sigma)\}$, the probabilistic reachability analysis of the network $\mathcal{F}$ is the process of computing the probabilistic output set of the network and its associate probability, i.e., $Y = \mathcal{F}(x), x \in X$.*

Definition 2.2 (Quantitative Verification). *Given a ReLU network $\mathcal{F}$, a constrained probabilistic input set $X = \{x \in \mathbb{R}^n | \, Cx \leq d \ \wedge \ x \sim \mathcal{N}(\mu, \Sigma)\}$, and a linear specification $S \triangleq Hy \leq g$, where y is the output of the network, the quantitative verification of the network is the process of computing the probability of the network satisfying its specification, i.e., $\mathcal{P}(Hy \leq g, y \in Y = \mathcal{F}(x), x \in X)$, where $\mathcal{P}$ states for a probability.*

In the following section, we introduce a new concept of probabilistic star set, its properties, and probability computation, which are important to solve the probabilistic reachability and quantitative verification of a ReLU network.

3 PROBABILISTIC STAR

3.1 Definition and properties

Definition 3.1 (Probabilistic Star). *A probabilistic star (or simply probstar) Θ is a tuple $\langle c, V, \mathcal{N}, P, l, u\rangle$ where $c \in \mathbb{R}^n$ is the center, $V = \{v_1, v_2, \cdots, v_m\}$ is a set of m vectors in $\mathbb{R}^n$ called basis vectors, $P : \mathbb{R}^m \rightarrow \{\top, \bot\}$ is a predicate, l and u are the lower-bound and upper-bound vectors of the predicate variables, which are random variables of a Gaussian distribution $\mathcal{N}$. The basis vectors are arranged to form the probstar's $n \times m$ basis matrix. The set of states represented by the probstar is given as:*

$$\begin{gathered}[\![\Theta]\!] = \{x \mid x = c + \Sigma_{i=1}^{m}(\alpha_i v_i), \ \alpha = [\alpha_1, \cdots, \alpha_m]^T \sim \mathcal{N}, \\ P(\alpha) \triangleq C\alpha \leq d, \ l[i] \leq \alpha_i \leq u[i], \}.\end{gathered} \tag{1}$$

We will refer to both the tuple Θ and the set of states $[\![\Theta]\!]$ as Θ.

Definition 3.2 (Probability). *Given a probstar Θ, the probability of the probstar is the probability of the predicate random variables $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_m]^T$ satisfying its constraints and bounds, i.e., $\mathcal{P}(\Theta) = \mathcal{P}(C\alpha \leq d \wedge \ l \leq \alpha \leq u, \ \alpha \sim \mathcal{N}(\mu, \Sigma))$. A probstar is an empty set if its probability is zero, i.e., $\mathcal{P}(\Theta) = 0$.*

Definition 3.3 (Boundness). *A probstar $\Theta = \langle c, V, \mathcal{N}, P, l, u\rangle$ is bounded if the predicate $P(\alpha) \triangleq C\alpha \leq d$ a bounded polytope.*

Proposition 1. *Any bounded constrained probabilistic input set $X \triangleq \{x \mid Cx \leq d \wedge x \sim \mathcal{N}(\mu, \Sigma), x \in \mathbb{R}^n\}$ is a probstar.*

Proposition 2 (Affine Mapping). *Given a probstar set $\Theta = \langle c, V, \mathcal{N}, P, l, u\rangle$, an affine mapping of the probstar Θ with the mapping matrix W and offset vector b defined by $\bar{\Theta} = \{y \mid y = Wx +$*

$b,\ x \in \Theta\}$ *is another probstar with the following characteristics:* $\bar{\Theta} = \langle \bar{c}, \bar{V}, \bar{\mathcal{N}}, \bar{P}, \bar{l}, \bar{u} \rangle,\ \bar{c} = Wc + b,\ \bar{v} = \{Wv_1, Wv_2, \cdots, Wv_m\},$ $\bar{\mathcal{N}} = \mathcal{N}, \bar{P} \equiv P, \bar{l} \equiv l, \bar{u} \equiv u.$

Proposition 3 (Intersection). *The intersection of a probstar* $\Theta \triangleq \langle c, V, \mathcal{N}, P, l, u \rangle$ *and a half-space* $\mathcal{H} \triangleq \{x \mid Hx \leq g\}$ *is another probstar with the following characteristics.*

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

Proposition 4 (Optimized range of a state). *Given a probstar set* $\Theta = \langle c, V, \mathcal{N}, P, l, u \rangle$*, the range of the* i^{th} *state* $x[i]$ *of the probstar set can be found by solving the following linear programming optimization problems:*

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

Proposition 5 (Estimated range of a state). *Given a probstar set* $\Theta = \langle c, V, \mathcal{N}, P, l, u \rangle$*, the range of the state vector* x *of the probstar set can be estimated quickly without solving the linear programming optimization problems by using only the lower bound and upper bound vectors of the predicate variables.*

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

3.2 Computing a probstar's probability

The ultimate goal of this work is to compute the probability of a network violating its specification given a constrained, probabilistic input set represented as a probstar. Therefore, one key component in our approach is the computation of the probability of a probstar, which is an *extension* of the following m-dimensional multivariate normal law under linear restrictions [6, 12]:

$$f(x) = \frac{1}{p} exp(-\frac{1}{2}x^T x)\mathbb{I}\{\underline{d} \leq Cx \leq \bar{d}\},$$
$$x = [x_1, \ldots, x_m]^T, C \in \mathbb{R}^{h \times m}, \bar{d}, \underline{d} \in \mathbb{R}^h, \tag{2}$$

where $\mathbb{I}\{\cdot\}$ is the indicator function, $rank(C) = h \leq m$, and $p = \mathcal{P}(\underline{d} \leq Cx \leq \bar{d})$ is the probability that a random vector x with standard normal (Gaussian) distribution in d-dimensions, i.e., $x \sim \mathcal{N}(0, I_m)$, falls in the H-polytope defined by the linear inequalities, where I_m is a m-dimensional identity matrix.

Although there are efficient methods [6, 13] for computing the probability $\mathcal{P}(\underline{d} \leq Cx \leq \bar{d})$, these methods require the constraint matrix A be a full rank matrix, $rank(C) = h$ and $h \leq m$, which is not usually the case in our problem. After the ReLU activating operation in reachability analysis, new constraints will be added to a probstar input set, making the predicate constraint matrix, i.e., C, become a rank-deficient matrix, i.e., $rank(C) < h$, and $h > m$ indicating that number of linear constraints is larger than the number of predicate random variables in general. In this section, we propose an approach for handling this general case using *singular value decomposition* (SVD) and *Gaussian approximation of a Dirac Delta distribution.*

Gaussian approximation of a Dirac Delta distribution. We use a simple example in the following to illustrate the idea. Let's consider a 2-dimensional Gaussian distribution $\mathcal{N}$:

$$\alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} \sim \mathcal{N}(\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \Sigma = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}).$$

This distribution is singular as the variance matrix Σ is not a positive definite matrix. We can see that the distribution $\mathcal{N}$ is a projection of a non-singular 1-dimensional Gaussian distribution $\mathcal{N}_2(0, 1)$ onto a hyperplane $\alpha_1 = 0$ in a 2-dimensional space (α_1, α_2). In other words, the probability mass of the distribution $\mathcal{N}$ relies completely on hyperplane $\alpha_1 = 0$. If we want to compute the probability $\mathcal{P}(2\alpha_1 + \alpha_2 \leq 2)$, the well-known Genz [13] and Botev [6] methods do not work as the distribution is singular. However this probability can be obtained manually as follows, $\mathcal{P}(2\alpha_1 + \alpha_2) \leq 2 = \mathcal{P}(\alpha_2 \leq 2) = 0.9772$ as $\alpha_1 = 0$. The question is how we can compute this probability automatically using the Genz and Botev methods. Remind that the first random variable belongs to a normal distribution with zero variance $\alpha_1 \sim \mathcal{N}_1(0, 0)$. This means that its associated probability density function (PDF) is a Dirac Delta function. Let's consider a Gaussian distribution with a PDF $\delta_\epsilon(x) = \frac{1}{|\epsilon|\sqrt{\pi}} e^{-(\frac{x}{\epsilon})^2}$. We have $\delta(x) = \lim_{\epsilon \to 0} \delta_\epsilon(x)$, indicating that a Direct Delta distribution can be approximated by a Gaussian distribution with a tiny standard deviation $\epsilon \approx 0$. Therefore, the singular Gaussian distribution $\mathcal{N}$ can be approximated by a non-singular Gaussian distribution $\mathcal{N}'$ as follows.

$$\alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} \sim \mathcal{N}'(\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \Sigma = \begin{bmatrix} \epsilon^2 & 0 \\ 0 & 1 \end{bmatrix}).$$

where ϵ is a very small positive value. Let $\epsilon = 10^{-4}$, the probability $\mathcal{P}(2\alpha_1 + \alpha_2 \leq 2),\ [\alpha_1, \alpha_2]^T \sim \mathcal{N}'$ can be computed using the Genz [13] and Botev [6] methods with the result of 0.9772.

Algorithm for computing a probstar's probability. From Definition 3.2, a probstar's probability is:

$$\mathcal{P}(\Theta) = \mathcal{P}(C\alpha \leq d \wedge\ l \leq \alpha \leq u,\ \alpha \sim \mathcal{N}(\mu, \Sigma))m$$
$$= \mathcal{P}(C'\alpha \leq d',\ \alpha \sim \mathcal{N}(\mu, \Sigma)),$$
$$C' = \begin{bmatrix} C; & I_m; & -I_m \end{bmatrix}, d' = \begin{bmatrix} d; & u; & -l \end{bmatrix}.$$

We consider the general case where C' is not a full row rank matrix that the existing methods cannot be applied directly to compute the probability. Assume that $C' \in \mathbb{R}^{h \times m}, d' \in \mathbb{R}^h$, where $h \gg m$. Let $r = rank(C') \leq m$. Using SVD decomposition, we have:

$$C' = UQV^T, U \in \mathbb{R}^{h \times h}, Q \in \mathbb{R}^{h \times m}, V \in \mathbb{R}^{m \times m},$$
$$Q = \begin{bmatrix} S & 0_{r \times (m-r)} \\ 0_{(h-r) \times r} & 0_{(h-r) \times (m-r)} \end{bmatrix}, S = \begin{bmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_r^2 \end{bmatrix} \in \mathbb{R}^{r \times r}$$
$$V^T = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix}, V_{11} \in \mathbb{R}^{r \times r}, V_{12} \in \mathbb{R}^{r \times (m-r)},$$
$$V_{21} \in \mathbb{R}^{(m-r) \times r}, V_{22} \in \mathbb{R}^{(m-r) \times (m-r)}.$$

Let $\bar{Q} = QV^T$, $\alpha' = \bar{Q} \times \alpha$, we can see that $\alpha' \in \mathbb{R}^h$ has the form of:

$$\alpha' = \begin{bmatrix} S' \\ 0_{(h-r) \times m} \end{bmatrix} \alpha,\ S' = [SV_{11}\ SV_{12}] \in \mathbb{R}^{r \times m}$$

In other words, α' is an affine mapping of α from m-dimensional space to h-dimensional space. As $\alpha \sim \mathcal{N}(\mu, \Sigma)$, α' is a vector of random variables of a singular Gaussian distribution $\mathcal{N}'$ whose the distribution mass relies on the hyper plane $\alpha'_{r+1} = \cdots = \alpha'_h = 0$. As shown before, we can approximate a singular Gaussian distribution by a non-singular Gaussian distribution. Let $a = S'\alpha \sim \mathcal{N}(S'\mu, S'\Sigma(S')^T), b \sim \mathcal{N}(0_{h-r}, \epsilon I_{h-r})$, we have:

$$\alpha' \approx [a\ b]^T \sim \mathcal{N}(\mu', \Sigma')$$

$$\mu' = \begin{bmatrix} S'\mu \\ 0_{h-r} \end{bmatrix},\ \Sigma' = \begin{bmatrix} S'\Sigma(S')^T & 0 \\ 0 & \epsilon I_{h-r} \end{bmatrix}$$

Note that U is an orthogonal matrix and thus is full row rank. Therefore, we can compute $\mathcal{P}(\Theta) = \mathcal{P}(U\alpha' \le d', \alpha' \sim \mathcal{N}(\mu', \Sigma')$ using existing methods [6, 13]. Algorithm 1 summarizes the computation of a probstar's probability.

Algorithm 1 Computation of a probstar's probability

Input: $\Theta = \langle c, V, \mathcal{N}, P, l, u\rangle$ # A probstar
Output: p # probability

procedure p = getProbability($\Theta = \langle c, V, \mathcal{N}, P, l, u\rangle$)
  $C \leftarrow P.C, d \leftarrow P.d, \mu \leftarrow \mathcal{N}.\mu, \Sigma \leftarrow \mathcal{N}.\Sigma$
  $C' = \begin{bmatrix} C & I_m & -I_m \end{bmatrix}^T, d' = \begin{bmatrix} d & u & -l \end{bmatrix}^T$
  $U, S, V^T = SVD(C')$ # SVD decomposition
  $V^T = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix}, V_{11} \in \mathbb{R}^{r\times r}, V_{12} \in \mathbb{R}^{r\times(m-r)}$
  $S' = [SV_{11}\ SV_{12}] \in \mathbb{R}^{r\times m}$
  $\mu' = \begin{bmatrix} S'\mu \\ 0_{h-r} \end{bmatrix},\ \Sigma' = \begin{bmatrix} S'\Sigma(S')^T & 0 \\ 0 & \epsilon I_{h-r} \end{bmatrix}$
  return $p = \mathcal{P}(U\alpha' \le d', \alpha' \sim \mathcal{N}(\mu', \Sigma')$ # using [6, 13]

4 PROBABILISTIC STAR REACHABILITY

Based on the proposed probabilistic star set representation, we can perform quantitative verification for ReLU networks using reachability analysis. Our verification approach has two steps: 1) computing the probabilistic reachable set of a ReLU network's output (reachability analysis) and 2) computing the probability of the output reachable set violating a given user-defined specification.

4.1 Probabilistic Reachable set computation

Computing the reachable set of a ReLU network can be done layer-by-layer. In this approach, a probstar input set can be propagated through the network layer-by-layer to construct the output reachable set. The core computation reduces to a single-layer reachability problem which can be done in two steps: *affine mapping* and *ReLU activating*. Let Θ be a probstar input set to a layer L with weight matrix W and bias vector b, the output reachable set R of the layer is:

$$\Theta = \langle c, V, \mathcal{N}, P, l, u\rangle,$$
$$\Theta' = W\Theta + b = \langle Wc + b, WV, \mathcal{N}, P, l, u\rangle,$$
$$R = L(\Theta) = ReLU(W\Theta + b) = ReLU(\Theta').$$

Similar to star reachability [30], *ReLU activating* on a probstar set, i.e., $ReLU(\Theta')$, can be done neuron-by-neuron by a sequence of stepReLU operations $ReLU_i, i = 1, 2, \ldots, n$, where n is the number of neurons in the layer.

$$R = ReLU_n((ReLU_{n-1}(\ldots(ReLU_1(\Theta'))))).$$

A stepReLU operation $ReLU_i$ applies the ReLU activation on the i^{th} input x_i of the input set. Depending on the range of x_i, an input set can be split into two intermediate sets after a stepReLU operation. Let lb_i and ub_i be the lower bound and upper bound of the input x_i, i.e., $lb_i \le x_i \le ub_i$. To compute the intermediate reachable set in a stepReLU operation, we consider three cases. First, if $lb_i \ge 0$, we have $x_i \ge 0$, thus $ReLU(x_i) = x_i$. Second, if $ub_i \le 0$, we have $x_i \le 0$, thus $ReLU(x_i) = 0$. Third, if $lb_i < 0$ and $ub_i > 0$, the input x_i can be larger or smaller than zero. Therefore, applying the ReLU activation function produces two possible outputs: 0 when $x_i \le 0$ or x_i when $x_i > 0$. Figure 1 illustrates a stepReLU operation on the first input x_1 in a 2-dimensional input set Θ' in which $lb_1 < 0$ and $ub_1 > 0$. In this example, the input set is split into two sets

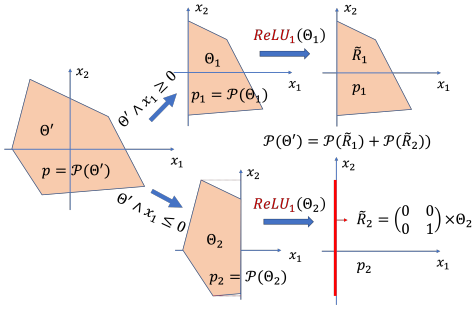


Figure 1: stepReLU operation on a probstar.

$\Theta_1 = \Theta' \wedge x_1 \ge 0$ and $\Theta_2 = \Theta' \wedge x_1 \le 0$. As we known for all $x \in \Theta_1, x_1 \ge 0$, we have $\tilde{R}_1 = ReLU_1(\Theta_1) = \Theta_1$. Similarly, for all $x = [x_1, x_2]^T \in \Theta_2,\ x_1 \le 0$, thus $\tilde{R}_2 = ReLU_1(\Theta_2) = [0,\ x_2]^T$. The visualization of the intermediate reachable sets $\tilde{R}_1, \tilde{R}_2$ is shown in Figure 1.

Since after one stepReLU operation, an input set can be split into two new intermediate sets, the output reachable set of an n-neurons ReLU layer may contain 2^n probstars in the worst case. One can see that the intersection (Proposition 3) and affine mapping (Proposition 2) are two basic computation steps involved in a stepReLU operation. These operations do not change the internal Gaussian distribution of the predicate variables. They only change the probability of the obtained intermediate reachable sets compared to the input set's probability. Importantly, as $\Theta = \Theta_1 \cup \Theta_2$, the sum of all intermediate reachable sets' probability equals the input set's probability, i.e., $\mathcal{P}(\Theta') = \mathcal{P}(\tilde{R}_1) + \mathcal{P}(\tilde{R}_2)$.

4.2 Reducing reachability time

To perform the stepReLU operation, we need to know the lower bound lb_i and upper bound ub_i of input x_i that can be obtained by solving linear programming optimization problems (Proposition 4), which are costly. Fortunately, we can significantly reduce the number of LPs solved in reachability by estimating these bounds quickly using the predicate variables' bounds (Proposition 5) without solving LPs. If the estimated lower bound $lb_i^{est} \ge 0$, we can move to the next stepReLU operation quickly without doing anything. If the estimated upper bound $ub_i^{est} \le 0$, we can construct the intermediate reachable set by projecting (affine mapping operation) the input set onto the hyperplane $x_i = 0$. If $lb_i^{est} < 0$ and $ub_i^{est} > 0$,

a split may occur at input x_i. In this case, we can solve LPs to determine if the split actually occurs to construct the intermediate reachable sets.

Updating predicate variables' bounds. Estimating *tight bounds* of the input x is important to reduce the number of LPs solved in reachability analysis. When a probstar is split, e.g., $\Theta = \Theta_1 \cup \Theta_2$ in Figure 1, a new constraint is added to a probstar to form a new set, e.g., $\Theta_1 = \Theta \wedge x_1 \geq 0$. This constraint will be transformed into a new constraint on predicate variables (Proposition 3). Adding this new constraint may change the lower bound and upper bound vectors of the predicate variables. If we do not update new lower bound and upper bound vectors for the predicate variables α, the next estimation of the lower bound and upper vectors of inputs (to neurons in later layers) based on the old predicate variable's bounds will be more and more conservative and not useful. Therefore, it is important to update new predicate variables' bounds after adding a new constraint to a probstar. In this paper, we implement a lightweight domain contraction method to update these bounds [3].

The idea of updating predicate variables' bounds is illustrated by a small example in Figure 2. In this example, we have a 2-

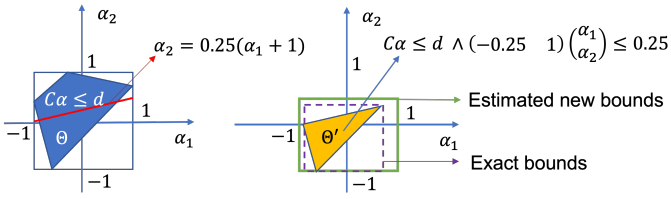


Figure 2: Updating predicate variables' bounds.

dimensional probstar Θ with predicate $P \triangleq C\alpha \leq d$, $\alpha = [\alpha_1, \alpha_2]^T$ and the predicate variables' bounds $l = [-1, -1]^T$, $u = [1, 1]^T$ (the blue box in the figure). If a new constraint $-0.25\alpha_1 + \alpha_2 \leq 0.25$ is added to this probstar, a new Θ' is obtained with a new predicate $P' \triangleq C\alpha \leq d \wedge (-0.25\alpha_1 + \alpha_2 \leq 0.25)$. One can see that the *exact new bounds* of the predicate variables in Θ' (the purple rectangle in Figure 2) can be obtained by solving four LPs, i.e., $min(max)\alpha_i,\ i = 1, 2,\ s.t.\ P' \triangleq C\alpha \leq d \wedge (-0.25\alpha_1 + \alpha_2 \leq 0.25)$. However, as we want to reduce the reachability time by avoiding solving LPs, we estimate the new bounds of the predicate variables (the green rectangle) using only their old bounds (the blue rectangle) and the new constraint. From the new constraint and the old bounds, we have: $\alpha_2 \leq 0.25 + 0.25\alpha_1 \leq 0.25 + 0.25 = 0.5$. One can see that in the new probstar Θ', the predicate variable α_2 has a new upper bound of 0.5 instead of 1 as in the original probstar Θ.

4.3 Quantitative reachability algorithm

The quantitative reachability algorithm for computing the probstar reachable sets of a layer and their associated probability is presented in Algorithm 2. The algorithm performs the affine mapping operation on the probstar input set Θ using the layer's weight matrix W and bias vector b (line 2). The obtained set Θ' will be the input to a sequence of stepReLU operations $ReLU_i$ (lines 3 to 10). Remind that a stepReLU operation may split a set into two new sets. Therefore, a for loop is used to handle multiple inputs at a specific stepReLU operation (lines 8, 9). After constructing the reachable sets of the layer (line 10), we compute their associated probability using Algorithm 1 (lines 12 to 14).

Algorithm 2 Quantitative reachability for a ReLU layer.

Inputs: $L = (W, b), \Theta = \langle c, V, N, P, l, u\rangle$ # A ReLU layer, a probstar input set
Output: R # A tuple of reachable sets and their associated probabilities

1: **procedure** p = quantitativeLayerReach(L, Θ)
2: $\Theta' = \Theta.affineMap(W, b)$ # affine mapping (Proposition 2)
3: $n = L.numberOfNeurons$ # number of neurons
4: $\tilde{R} = \Theta'$
5: **for** i from 1 to n **do**
6: $m = length(\tilde{R})$ # number of probstars
7: $O = [\,]$
8: **for** j from 1 to m **do** # handling multiple inputs at $ReLU_i$
9: $O \leftarrow ReLU(\tilde{R}_j, i)$
10: $\tilde{R} = O$
11: $m = length(\tilde{R}), R = ()$
12: **for** j from 1 to m **do**
13: $p_j = getProbability(\tilde{R}_j)$ # Algorithm 1
14: $R \leftarrow (\tilde{R}_j, p_j)$
15: return R
16: **procedure** $\tilde{R}$ = ReLU($\tilde{\Theta}, i$) # stepReLU operation
17: $lb_i, ub_i = \tilde{\Theta}.estimateBounds(i)$ # Proposition 5
18: **if** $lb_i \geq 0$ **then**
19: $\tilde{R} = \tilde{\Theta}$
20: **if** $ub_i \leq 0$ **then**
21: $\tilde{R} = \tilde{\Theta}.resetRow(i)$ # $y[i] = ReLU(x[i]) = 0$
22: **if** $lb_i < 0$ & $ub_i > 0$ **then**
23: $x_{min} = \tilde{\Theta}.getMin(i)$ # get optimal lower bound (Proposition 4)
24: **if** $x_{min} \geq 0$ **then**
25: $\tilde{R} = \tilde{\Theta}$
26: **else if** $x_{min} < 0$ **then**
27: $x_{max} = \tilde{\Theta}.getMax(i)$ # get optimal upper bound (Proposition 4)
28: **if** $x_{max} \leq 0$ **then**
29: $\tilde{R} = \tilde{\Theta}.resetRow(i)$ # $y[i] = ReLU(x[i]) = 0$
30: **else**
31: $\tilde{R}_1 = \tilde{\Theta}.addConstraint(x[i] \geq 0)$ # Proposition 3
32: $\tilde{R}_1.updatePredicateVariableBounds()$
33: $\tilde{R}_2 = \tilde{\Theta}.addConstraint(x[i] \leq 0)$ # Proposition 3
34: $\tilde{R}_2.updatePredicateVariableBounds()$
35: $\tilde{R}_2.resetRow(i)$
36: $\tilde{R} \leftarrow [\tilde{R}_1, \tilde{R}_2]$
37: return $\tilde{R}$

Complexity and output reachable set probability. The following lemma extends the complexity and output reachable set probability of a layer to an N-neurons feed-forward neural network $\mathcal{F}$.

Lemma 6. *Given an N-neurons feed-forward ReLU neural network $\mathcal{F}$ and a probstar input set Θ, the exact output reachable set O of the network may contain multiple probstars $O = \{O_1, O_2, \ldots, O_M\}$ in which $M \leq 2^N$ and $\mathcal{P}(\Theta) = \mathcal{P}(O) = \mathcal{P}(O_1) + \mathcal{P}(O_2) + \cdots + \mathcal{P}(O_M)$.*

5 QUANTITATIVE VERIFICATION

Quantitative verification of ReLU networks can be done precisely or overapproximately. In the exact quantitative verification, we construct all the probstar output reachable sets of the network and compute the *exact total probability* p_v of the output reachable sets intersecting with the region defined by the specification. In addition, *complete set of counterexamples* C is constructed. In approximate quantitative verification, the users provide a probability

threshold p_f ($0 < p_f \ll 1$) to filter out all the reachable sets (at each layer) whose probabilities are smaller than the threshold. The total probability of neglected reachable sets is combined with the total probability of remaining output reachable sets violating the specification to obtain the *lower bound* $\underline{p}_v$ and *upper bound* $\bar{p}_v$ of the probability of the network violating its specification. Additionally, a *partial set counterexamples* may be obtained.

Handling the tail. Qualitative verification methods work only with bounded input sets. Unbounded input sets lead to unbounded output sets that are difficult and expensive to compute. In other words, computing exact output reachable sets of a ReLU network with unbounded input sets is computationally intractable in general. In our quantitative verification, we can provide a probabilistic guarantee for a neural network for the entire unbounded input space. We use a 1-dimensional example to illustrate the idea. In this example, suppose we are considering a bounded input set $X = \{x \mid \mu - 3\sigma \leq x \leq \mu + 3\sigma\}$. The probability of this input set is $\mathcal{P}(X) = 0.9973$. Assume that we verify a network $\mathcal{F}$ given the input set $\Theta \equiv X$, and a specification $spec \triangleq Hy \leq g$, and the quantitative verification algorithm returns the lower bound $\underline{p}_v$ and upper bound $\bar{p}_v$ of the probability of violation (note that in the exact verification, we have $\underline{p}_v = \bar{p}_v = p_v$). Now, what we want to know is the probability of violation for all $-\infty \leq x \leq \infty$ (i.e., entire unbounded input space). We can decompose the entire unbounded input space into two sets X and $\bar{X}$ (note that $\bar{X}$ is unbounded and non-convex). We have $\mathcal{P}(X) + \mathcal{P}(\bar{X}) = 1 \rightarrow \mathcal{P}(\bar{X}) = 1 - 0.9973 = 0.0027$. One can see that the maximum probability of the network $\mathcal{F}$ violating its specification given the input set $\bar{X}$ is $\mathcal{P}(\bar{X})$, i.e., all inputs in $\bar{X}$ make the network violate the specification. Therefore, the maximum probability of the network $\mathcal{F}$ violating its specification for all $-\infty \leq x \leq \infty$ is $\bar{p}_v + 1 - \mathcal{P}(X) = \bar{p}_v + 0.0027$. Similarly, the minimum probability of the network violating its specification is $\underline{p}_v$, i.e., optimistically, none of inputs inside $\bar{X}$ are counterexamples.

Quantitative verification algorithms. Our quantitative verification algorithm is described in Algorithm 3. If the filtering probability $p_f = 0$, then the algorithm is the exact quantitative verification. If $p_f > 0$, then the algorithm becomes the overapproximate quantitative verification. At each layer, the algorithm computes the probabilistic reachable sets (lines 6-8), filters out all reachable sets whose probabilities are smaller than p_f (line 12), and updates the total probability of filtered reachable sets (line 13). The reachable sets of the last layer are the output reachable sets of the network (line 14). The intersection of output reachable sets of the network with the specification and its probability are computed (lines 17, 18). If this probability is larger than zero, the total probability of violation is updated (line 19), and a counterexample set is conducted (lines 20, 21) similar to [30]. Additionally, the unsafe output set is stored (line 23). Then, we compute the lower bound and upper bound of the probability of violation (lines 22 to 25) and the minimum and maximum probability of violation when considering the entire infinite input space (lines 26). Finally, we return the output reachable sets, unsafe output sets, counterexample sets, and the bounds of the probability of violation (line 27).

Soundness and Completeness. The soundness and completeness of our quantitative verification algorithm are described in the following lemmas.

Algorithm 3 Quantitative verification of a ReLU network.

Inputs: $\mathcal{F} = \{L_i\}, \Theta, p_f$ # Network, input set, filtering probability
$spec = (H, g)$ # specification: $Hy \leq g$
Output: $\mathcal{R}, \tilde{\mathcal{R}}, C, \underline{p}_v, \bar{p}_v, p_v^{min}, p_v^{max}$ # Output reachable set,
unsafe output set, counterexample set, bounds of violating probability

1: **procedure** p_v, C = quantitativeVerify($\mathcal{F}, \Theta, p_f, spec$)
2: $n \leftarrow \mathcal{F}.numberOfLayers$
3: $I = \Theta$ # probstar input sets to each layer
4: $\mathcal{R} = ()$ # reachable sets and their associated probability
5: $p_{ignored} = 0$ # total probability of filtered reachable sets
6: **for** i from 1 to n **do** # layer reachability
7: $O_i = []$ # output set of i^{th} layer
8: **for** j from 1 to length(I) **do** # each layer process multiple inputs
9: $M = quantitativeLayerReach(L_i, I_j)$
10: **for** k from 1 to length(M) **do**
11: $(R_k, p_k) \leftarrow M[k]$
12: **if** $p_k \geq p_f$ **then** $\mathcal{R} \leftarrow (R_k, p_k)$, $O_i \leftarrow R_k$ # filtering
13: **else** $p_{ignored} = p_{ignored} + p_k$
14: $I = O_i$
15: $\tilde{\mathcal{R}} = []$
16: **for** i from 1 to length($\mathcal{R}$) **do**
17: $(R_i, p_i) \leftarrow \mathcal{R}[i]$
18: $\tilde{R}_i = R_i.addConstraints(Hy \leq g)$ # intersect with specification
19: $\tilde{p}_i = getProbability(\tilde{R}_i)$
20: **if** $\tilde{p}_i > 0$ **then** $p_v = p_v + \tilde{p}_i$ # update probability of violation
21: $C_i = \langle \Theta.c, \Theta.V, \Theta.\mathcal{N}, \tilde{R}_i.P, \tilde{R}_i.l, \tilde{R}_i.u \rangle$ # counterexample set
22: $C \leftarrow C_i$
23: $\tilde{\mathcal{R}} \leftarrow \tilde{R}_i$
24: **if** $p_f = 0$ **then**
25: $\bar{p}_v = p_v, \underline{p}_v = p_v$
26: **else**
27: $\underline{p}_v = p_v, \bar{p}_v = p_v + p_{ignored}$
28: $p_v^{min} = \underline{p}_v, p_v^{max} = \bar{p}_v + 1 - \mathcal{P}(\Theta)$
29: return $\mathcal{R}, \tilde{\mathcal{R}}, C, \underline{p}_v, \bar{p}_v, p_v^{min}, p_v^{max}$

Lemma 7 (Soundness). *Given a ReLU network $\mathcal{F}$, a bounded probstar input set Θ, and a specification $spec \triangleq Hy \leq g$, the probability of the network violating its specification for all inputs in the input set satisfies $\underline{p}_v \leq p \leq \bar{p}_v$, and the probability of the network violating its specification for entire input space satisfies $p_v^{min} \leq p \leq p_v^{max}$, where $\underline{p}_v, \bar{p}_v, p_v^{min}, p_v^{max}$ are the bounds obtained from Algorithm 3.*

Lemma 8 (Completeness). *The Algorithm 3 is complete when the exact verification scheme is used to verify a ReLU network with a given bounded probstar input set.*

Remark 5.1. *It is important to emphasize that the soundness and completeness of the proposed quantitative verification algorithm can be achieved under the assumption that the estimation of the probability of a probstar is precise (Algorithm 1). In practice, there is a very tiny error (which is ignored in the analysis) in the probability estimation caused by the Gaussian approximation of a Dirac Delta distribution and the existing **non-deterministic** techniques [6, 13].*

6 EVALUATION

Implementation. We implement our approach in a tool named ***StarV*** (Star-based Verification tool) using python3. We use the built-in Botev method [6] in the Scipy package to compute the probability of a probabilistic star. In StarV, we support parallel computation using the multiprocessing module to speed up the verification procedure. In this section, we first describe our approach in detail using a tiny network and then analyze its performance

on the well-known ACASXU networks [22], and the rocket lander benchmark for SpaceEx Falcon 9 [37]. Our experiment is done on a computer with the following configurations: Intel Core i7-10700 CPU @ 2.90GHz × 8 Processors, 63.7 GiB Memory, 64-bit Ubuntu 18.04.6 LTS OS.[1].

We will use the following notations for our experiments: p_f is the filtering probability, O is the number of output sets, $\mathcal{US} - O$ is the number of unsafe output sets, $US - Prob - LB$ and $US - Prob - US$ are the lower bound and upper bound of the probability of violation, $US - Prob - Min$ and $US - Prob - Max$ are the minimum, and maximum unsafe probability of the network for the entire infinite input space, $I - Prob$ is the input set's probability, VT is the verification time in second.

6.1 Verifying a tiny network

We use a tiny network to describe our approach. The tiny network has two inputs, two outputs, one ReLU layer, and one fully connected output layer. The weights and biases for the ReLU and output layers, respectively, are:

$$W_1 = \begin{bmatrix} 1 & -2 \\ -1 & 0.5 \\ 1 & 1.5 \end{bmatrix},\ b_1 = \begin{bmatrix} 0.5 \\ 1.0 \\ -0.5 \end{bmatrix},\ W_2 = \begin{bmatrix} -1 & -1 & 1 \\ 2 & 1 & -0.5 \end{bmatrix},\ b_2 = \begin{bmatrix} -0.2 \\ -1.0 \end{bmatrix}$$

The input $x = [x_1, x_2]^T$ to the network belongs to a 2-dimensional Gaussian distribution with the area of interest is $-2 \leq x_1 \leq 2$ and $-1 \leq x_2 \leq 1$, i.e., the input lower bound $lb_x = [-2, -1]^T$ and the input upper bound $ub_x = [2, 1]^T$. The mean of the Gaussian distribution is chosen as $\mu = (lb_x + ub_x)/2 = [0, 0]^T$. The standard deviation of the distribution $\sigma = [\sigma_1, \sigma_2]^T$ is chosen such that $\mu + a \times \sigma = ub_x \rightarrow \sigma = (ub_x - \mu)/a$, where a is a positive coefficient. When a increases, the probability of the inputs lying between their lower and upper bounds of interest increases (see Figure **??**). In this case study, we choose $a = 2.5$ and thus, $\sigma = [0.8, 0.4]^T$. The variance of the distribution is $\Sigma = diag(\sigma_1^2, \sigma_2^2) = diag(0.64, 0.16)$, where $diag$ states for a diagonal matrix. We are interested in verifying the network against its unsafe specification defined by $spec \triangleq y_1 \leq -2$.

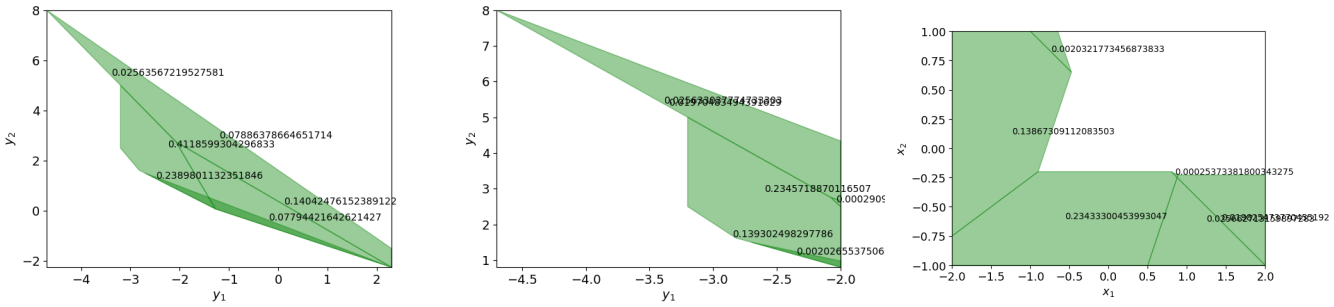
Compute and visualize output reachable sets. We represent the input set I as a probstar and propagate it through the network using Algorithm 2 to construct the probabilistic output reachable sets. The reachability time is approximately 0.09 seconds, and the output set consists 6 probstars, depicted in Figure 3a. The total probability of the output reachable sets equals the input set's probability $\mathcal{P}(I) \approx 0.975316$. **Verifying the network.** The verification results of the tiny network are presented in Table 1. We can see that the number of probstar output sets reduces when the filtering probability increases, i.e., p_f. With $p_f = 0.1$, the number of probstar output sets is 3 compared to 6 with exact verification ($p_f = 0$). In the exact verification, the lower bound and upper bound of unsafe probability ($US - Prob - LB$ and $US - Prob - UB$) are the same ($\underline{p}_v = \bar{p}_v = 0.420977$). This is also the minimum probability of violation for the entire input space, i.e., US-Prob-Min p_v^{min}. With $p_f = 0.1$, the estimated lower bound and upper bound of unsafe probability are 0.373799 and 0.556148. One can see that using the approximate verification scheme obtains a wider range of unsafe probability than the exact verification scheme. This is also true for the estimation of the minimum and maximum unsafe probability for the entire input space. As we filter out some reachable sets with small probabilities in verification, the total verification time (VT) is smaller than the one in the exact verification (0.127659 vs. 0.193592). Note that 4 cores are used for the verification of this case study. Our algorithm also returns the exact/partial unsafe output set and counterexample set, as depicted in Figures 3b and 3c (for exact verification).

[1]All experimental results are reproducible using StarV: https://github.com/V2A2/StarV

6.2 Verifying ACASXu Networks

The ACASXu networks [22] are the well-known benchmarks for comparing the performance of qualitative verification approaches. In this paper, we are interested in quantitative verification for ACASXu networks. We want to compute the probability of a network violating its properties. Therefore we consider properties (P_2, P_3, and P_4) and networks that return unsafe results in qualitative verification [3]. Detailed information on the properties and the networks can be found in [17, 22]. Similar to the tiny network case study, we use the lower bound lb and upper bound ub vectors of the inputs corresponding to each property to create probabilistic input sets for our quantitative verification. We choose the mean of the Gaussian distribution of the inputs as $\mu = (lb + ub)/2$ and the standard deviation $\sigma = (ub - \mu)/a$, where $a = 3.0$. Table 2 represents the quantitative verification results of unsafe ACASXu networks under different safety properties. The quantitative verification results for all networks can be found in Table 6 the Appendix.

Timing performance and conservativeness. Our quantitative verification algorithm successfully computes the bounds of the probability of violation of all unsafe networks (90 verification queries in total) in approximately 24 hours, with 8 cores used to speed up the verification time. Verification time varies for different networks and different properties. The maximum time for a verification query is less than 2 hours, while the minimum verification time is around 3 seconds. Table 2 shows that the approximate quantitative verification scheme can significantly reduce the verification time for a network with many output sets with wider probability bounds on the violation. For example, in property 2 and network 1-6, the total verification time of the exact scheme is 1424 seconds, while the approximate scheme finishes in 206.77 seconds ($\approx 7\times$ faster). In this case, the approximate scheme obtains a pretty tight lower bound of the probability of violation compared to the exact scheme ($2.80807e-06$ vs. $1.87224e-05$). However, its obtained upper bound of the probability of violation is conservative (0.0528349 vs. $1.87224e-05$). In the case that the number of output sets is small, the approximate scheme takes a longer time than the exact scheme, e.g, property 4 and network 1-9. This is because estimating the probabilities of intermediate probstars in reachability constitutes significance in the total verification time. Interestingly, for properties 3 and 4, the whole input sets associated with the properties are counterexample sets, i.e., all output sets violate the safety properties. We note that the estimation of probability bounds may change slightly time-to-time because of the non-determinism nature of current techniques [6, 13]. Therefore, it is not surprising that the repeatability of this work may produce slightly different results.



(a) Output set. **(b) Unsafe output set.** **(c) Counter input set.**

Figure 3: Tiny Networks' output set, unsafe output set, and counter input set.

p_f	$\mathcal{O}$	$\mathcal{US}-\mathcal{O}$	$\mathcal{C}$	US-Prob-LB	US-Prob-UB	US-Prob-Min	US-Prob-Max	I-Prob	VT
0	6	6	6	0.420977	0.420977	0.420977	0.445661	0.975316	0.193592
0.1	3	3	3	0.373799	0.556148	0.373799	0.580832	0.975316	0.127659

Table 1: Verification results of the tiny network.

Prop	Net	p_f	$\mathcal{O}$	$\mathcal{US}-\mathcal{O}$	$\mathcal{C}$	US-Prob-LB	US-Prob-UB	US-Prob-Min	US-Prob-Max	I-Prob	VT
2	1-6	0	376352	80621	80621	1.87224e-05	1.87224e-05	1.87224e-05	0.013445	0.986574	1424
2	1-6	1e-05	4953	2649	2649	2.80807e-06	0.0528349	2.80807e-06	0.0662612	0.986574	206.772
2	2-2	0	471882	320909	320909	0.0353886	0.0353886	0.0353886	0.0488149	0.986574	2102.47
2	2-2	1e-05	6093	4831	4831	0.0195646	0.0940341	0.0195646	0.10746	0.986574	298.999
2	2-9	0	909914	297873	297873	0.000997678	0.000997678	0.000997678	0.014424	0.986574	4561.2
2	2-9	1e-05	6895	4325	4325	0.000255108	0.106958	0.000255108	0.120384	0.986574	504.504
2	3-1	0	252573	78561	78561	0.0445325	0.0445325	0.0445325	0.0579588	0.986574	1086.38
2	3-1	1e-05	4753	2620	2620	0.0305	0.0726312	0.0305	0.0860575	0.986574	202.726
2	3-6	0	1003429	120329	120329	0.0335763	0.0335763	0.0335763	0.0470026	0.986574	5224.43
2	3-6	1e-05	5485	432	432	0.0207834	0.106866	0.0207834	0.120292	0.986574	451.958
2	3-7	0	475107	54475	54475	0.00404731	0.00404731	0.00404731	0.0174736	0.986574	2598
2	3-7	1e-05	5666	113	113	0.00231884	0.0749959	0.00231884	0.0884222	0.986574	331.1
2	4-1	0	402334	263892	263892	0.00231247	0.00231247	0.00231247	0.0157388	0.986574	1870.66
2	4-1	1e-05	5771	4743	4743	0.0010368	0.0716188	0.0010368	0.0850451	0.986574	305.28
2	4-7	0	651996	378735	378735	0.0409502	0.0409502	0.0409502	0.0543764	0.986574	3407.83
2	4-7	1e-05	5940	4476	4476	0.0207855	0.108069	0.0207855	0.121495	0.986574	418.933
2	5-3	0	125749	87	87	1.81747e-09	1.81747e-09	1.81747e-09	0.0134263	0.986574	418.844
2	5-3	1e-05	4130	7	7	1.58834e-09	0.032622	1.58834e-09	0.0460483	0.986574	139.698
3	1-7	0	500	500	500	0.976871	0.976871	0.976871	0.990298	0.986574	3.59073
3	1-7	1e-05	190	190	190	0.980113	0.980432	0.980113	0.993858	0.986574	4.75383
4	1-9	0	471	471	471	0.989244	0.989244	0.989244	1	0.989244	3.41508
4	1-9	1e-05	142	142	142	0.979634	0.980022	0.979634	0.990777	0.989244	3.6162

Table 2: Quantitative verification results for unsafe ACASXu networks.

Comparing with a sampling-based method. We compared our method with a simple Monte Carlo (MC) approach with different sample sizes on the network 1-2 for property 2. The MC approach obtains very different unsafe probabilities for different sample sizes. When the number of samples is small (N < 10,000), the MC can not find counterexamples which results in zero probability of violation. When N is larger, the MC method obtains larger unsafe probabilities compared to our result. For example, when $N = 10^5$, the unsafe probability obtained by the MC approach is 6×10^{-5} which is much larger than our result (1.3925×10^{-5}) (see appendix). To make the MC method result converge to our exact result, a very large number of samples is required, which increases the verification time significantly. In our experiment, when the sample size = 10^8, the program got killed due to a memory issue. We can see that our approach significantly outperforms the simple MC approach.

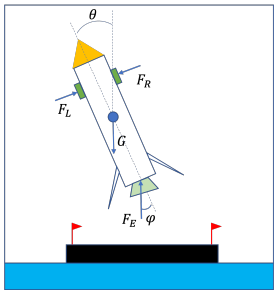


Figure 4: Rocket lander benchmark.

6.3 Verifying the rocket lander controllers

The rocket lander benchmark, shown in Figure 4, is a vertical rocket landing model simulating SpaceX's Falcon 9 first-stage rocket. It is based on the lunar lander presented in [7]. The rocket is controlled by the main engine thruster at the bottom with an actuated joint and two nitrogen thrusters attached to the top sides. The main thruster

produces power F_E in the range of $[0, 1]$ with a relative angle to the rocket body φ. The side thrusters produce power F_S in the range of $[-1, 1]$ in which -1 indicates that the right thruster has full throttle and the left thruster is turned off while 1 indicates the opposite. The goal is to control the rocket from a certain height to land in the center of the barge (the black rectangle) on the sea (the blue rectangle) without falling or crashing. To do that, we need to control the rocket's velocity and the lateral angle θ through the thrusters. There are three actions that need to be learned, the main engine thruster F_E, its angle φ, and the side nitrogen thrusters F_S. These actions are learned using reinforcement learning based on the observation of the position x, y of the rocket relative to the landing center on the barge, the velocity v_x and v_y of the rocket, its lateral angle θ, its angular velocity ω and the last three actions advisory F'_E, F'_S, φ',, i.e., the observation state vector is $[x, y, v_x, v_y, \theta, \omega, F'_E, F'_S, \varphi']^T$. Several agents are learned to control the rocker using the Deep Deterministic Policy Gradient (DDPG) [19] approach, which combines the Q-learning with Policy gradients. The learned agents are feedforward neural networks with 9 inputs, 3 outputs, and 5 hidden layers with 20 ReLU neurons per layer. Two safety properties are defined for the agents in the following [37]. The initial conditions associated with the safety properties are presented in the appendix.

- ***Property*** P_1: When $-20^\circ \leq \theta \leq -6^\circ, \omega < 0, \varphi' \leq 0, F'_S \leq 0$, the desired action should be $\varphi < 0$ or $F_S < 0$, which is the scenario where the agent should always prevent the rocket from tilting to the right.
- ***Property*** P_2: When $6^\circ \leq \theta \leq 20^\circ, \omega \geq 0, \varphi' \geq 0, F'_S \geq 0$, the desired action should be $\varphi > 0$ or $F_S > 0$, which is the scenario where the agent should always prevent the rocket from tilting to the left.

The verification results for two rocket networks are presented in Table 3. One can see that the two networks do not violate the second safety property, i.e., P_2. However, they both violate the first safety property with a very small probability. As the input space is large compared to the ACASXu case study, the exact verification scheme produces a very large number of output sets and has a pretty large verification time, e.g., the net 0 under the second property produces more than 2 million probstars output sets, but none of them violate the safety property.

7 RELATED WORKS

Qualitative Verification of DNNs. Qualitative verification of DNNs is an active research area. Notable sound and complete verification methods include Satisfiability Modulo Theory (SMT) [16, 17], optimization using mixed integer linear programming encoding [21], star-based [2, 30, 33], and facet-vertex incidence matrix [36], symbolic interval [35]. Although these methods provide both soundness and completeness, they are usually slow and not scalable to deal with large networks and large input space. Many over-approximate verification approaches have been proposed to overcome this drawback, such as MILP [9], semidefinite programming [11], abstract interpretation [24, 27, 28, 34, 39], overapproximate star reachability [29, 31], input quantization [15], and relaxed convex program [18], just to name a few.

Quantitative verification of DNNs. Quantitative verification of DNNs is an important topic that has not been explored much in the verification community. A few research papers focused on binary neural networks with quantized discrete input space [5, 26, 40]. Working with quantized discrete input spaces and sign activation functions makes quantitative verification less challenging, as the number of violations of network outputs can be counted. To the best of our knowledge, there is only one quantitative verification for ReLU DNNs with continuous input space [10]. In this work, the authors use an ellipsoid to represent the input set with Gaussian random variables. This ellipsoidal input set is propagated through the network to construct an overapproximate output ellipsoid with computable confidence by relaxing the ReLU activation functions using affine and quadratic constraints. In our approach, we consider a general input space with linear constraints on Gaussian random variables. We propose an approach to precisely propagate the polytope probabilistic input set through the network without any relaxation for the ReLU activation function. Due to this, we can accurately estimate the lower bound and upper bound of the probability of a ReLU network violating its property. Interestingly, the approach proposed in this paper was inspired by the recent work in learning the density distribution of autonomous systems [23]. However, instead of quantifying the risk of a black-box autonomous system, we focus on quantifying the risk of deep ReLU networks.

In [14], the authors proposed a verification approach for neural network control systems (NNCSs) using a custom neural network architecture. This architecture encoded an associated signal temporal logic formula as part of the network, allowing reachability analysis methods to be utilized for computing robustness of the formula. We emphasize that our work can be extended to NNCSs, with the advantage that the NN architecture does not need to be modified to a specific form. Furthermore, our probabilistic treatment of inputs enables reasoning over statistics such as risk of failure, which is not possible in the deterministic setting or without using sampling [38].

8 CONCLUSION

This paper proposed a new quantitative verification approach for ReLU networks using the probabilistic star concept, a new variant of the well-known star set. Exact and approximate verification schemes have been proposed and evaluated using the well-known ACASXu networks and recent rocket landing benchmarks. The experiments show that our approach is promising for verifying neural network controllers in the real world at design time. In the future, we will implement the proposed approach in a depth-first search manner [2] to reduce memory consumption in the analysis. More importantly, based on the concept of probstar, we will develop ProbStar temporal logic, a new specification language for specifying the temporal behavior of autonomous systems that is suitable for quantitative verification using star reachability.

REFERENCES

[1] Stanley Bak and Parasara Sridhar Duggirala. 2017. Rigorous simulation-based analysis of linear hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer.

[2] Stanley Bak and Parasara Sridhar Duggirala. 2017. Simulation-Equivalent Reachability of Large Linear Systems with Inputs. In *Proceedings of the 29th International Conference on Computer Aided Verification*. Springer.

Net-ID	Prop	p_f	$\mathcal{O}$	$\mathcal{US}-\mathcal{O}$	$\mathcal{C}$	US-Prob-LB	US-Prob-UB	US-Prob-Min	US-Prob-Max	I-Prob	VT
0	P_1	0	1018256	1674	1674	7.97799e-08	7.97799e-08	7.97799e-08	0.0240375	0.975963	5903.69
0	P_1	1e-08	82277	203	203	5.84833e-08	0.000228337	5.84833e-08	0.0242657	0.975963	5407.88
0	P_1	1e-05	7889	6	6	4.15404e-09	0.0674751	4.15404e-09	0.0915125	0.975963	1158.58
0	P_1	0.001	122	0	0	0	0.686172	0	0.710209	0.975963	147.821
0	P_2	0	2317149	0	0	0	0	0	0.0240374	0.975963	13132.7
0	P_2	1e-08	148357	0	0	0	0.000442907	0	0.0244803	0.975963	11183.7
0	P_2	1e-05	9929	0	0	0	0.105315	0	0.129352	0.975963	2216.01
0	P_2	0.001	99	0	0	0	0.803244	0	0.827281	0.975963	215.184
1	P_1	0	995322	1732	1732	8.6867e-08	8.6867e-08	8.6867e-08	0.0240375	0.975963	5163.86
1	P_1	1e-08	80202	18	18	5.14548e-08	0.00022693	5.14548e-08	0.0242643	0.975963	5753.19
1	P_1	1e-05	6107	0	0	0	0.0536327	0	0.0776701	0.975963	1229.73
1	P_1	0.001	149	0	0	0	0.529162	0	0.553199	0.975963	153.337
1	P_2	0	312129	0	0	0	0	0	0.0240374	0.975963	1495.59
1	P_2	1e-08	22876	0	0	0	6.00143e-05	0	0.0240974	0.975963	1613.42
1	P_2	1e-05	2798	0	0	0	0.0161751	0	0.0402125	0.975963	448.471
1	P_2	0.001	175	0	0	0	0.333419	0	0.357456	0.975963	109.729

Table 3: Quantitative verification results for rocket neural network controllers.

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

[4] Stanley Bak, Hoang-Dung Tran, and Taylor T. Johnson. 2019. Numerical Verification of Affine Systems with Up to a Billion Dimensions. In *Proceedings of the 22Nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC '19)*. ACM, New York, NY, USA, 23–32. https://doi.org/10.1145/3302504.3311792

[5] Teodora Baluta, Shiqi Shen, Shweta Shinde, Kuldeep S Meel, and Prateek Saxena. 2019. Quantitative verification of neural networks and its security applications. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1249–1264.

[6] Zdravko I Botev. 2017. The normal law under linear restrictions: simulation and estimation via minimax tilting. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 79, 1 (2017), 125–148.

[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

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

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

[10] Mahyar Fazlyab, Manfred Morari, and George J Pappas. 2019. Probabilistic verification and reachability analysis of neural networks via semidefinite programming. In *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, 2726–2731.

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

[12] Alan Genz and Frank Bretz. 2009. *Computation of multivariate normal and t probabilities*. Vol. 195. Springer Science & Business Media.

[13] Alan Genz and Giang Trinh. 2016. Numerical computation of multivariate normal probabilities using bivariate conditioning. In *Monte Carlo and Quasi-Monte Carlo Methods*. Springer, 289–302.

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

[15] Kai Jia and Martin Rinard. 2021. Verifying low-dimensional input neural networks via input quantization. In *International Static Analysis Symposium*. Springer, 206–214.

[16] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 97–117.

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

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

[19] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[20] Changliu Liu, Tomer Arnon, Christopher Lazarus, Clark Barrett, and Mykel J Kochenderfer. 2019. Algorithms for verifying deep neural networks. *arXiv preprint arXiv:1903.06758* (2019).

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

[22] Guido Manfredi and Yannick Jestin. 2016. An introduction to ACAS Xu and the challenges ahead. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. IEEE, 1–9.

[23] Yue Meng, Dawei Sun, Zeng Qiu, Md Tawhid Bin Waez, and Chuchu Fan. 2022. Learning density distribution of reachable states for autonomous systems. In *Conference on Robot Learning*. PMLR, 124–136.

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

[25] Sanjit A Seshia, Dorsa Sadigh, and S Shankar Sastry. 2022. Toward verified artificial intelligence. *Commun. ACM* 65, 7 (2022), 46–55.

[26] Andy Shih, Adnan Darwiche, and Arthur Choi. 2019. Verifying Binarized Neural Networks by Angluin-Style Learning.. In *SAT*. 354–370. https://doi.org/10.1007/978-3-030-24258-9_25

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

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

[29] Hoang-Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T Johnson. 2020. Verification of deep convolutional neural networks using imagestars. In *International conference on computer aided verification*. Springer, 18–42.

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

[31] Hoang-Dung Tran, Neelanjana Pal, Patrick Musau, Xiaodong Yang, Nathaniel P. Hamilton, Diego Manzanas Lopez, Stanley Bak, and Taylor T. Johnson. 2021. Robustness Verification of Semantic Segmentation Neural Networks using Relaxed Reachability. In *33rd International Conference on Computer-Aided Verification (CAV)*. Springer.

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

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

[34] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*. 6369–6379.

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

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

[37] Xiaodong Yang, Tom Yamaguchi, Hoang-Dung Tran, Taylor T. Johnson, and Danil Prokhorov. 2022. Neural Network Repair with Reachability Analysis. In *20th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*.

[38] Mojtaba Zarei, Yu Wang, and Miroslav Pajic. 2020. Statistical verification of learning-based cyber-physical systems. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*. 1–7.

[39] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. 2018. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems*. 4944–4953.

[40] Yedi Zhang, Zhe Zhao, Guangke Chen, Fu Song, and Taolue Chen. 2021. BDD4BNN: a BDD-based quantitative analysis framework for binarized neural networks. In *International Conference on Computer Aided Verification*. Springer, 175–200.

APPENDIX

Lemma 7's proof

Proof. When the exact verification scheme is used, i.e., $p_f = 0$, we have $\underline{p}_v = \bar{p}_v$ representing the exact total probability of the output reachable sets violating the specification given the bounded input set Θ. When the approximate verification scheme is used, we filter out some reachable sets in the analysis at each layer. In the worst case, all filtered reachable sets lead to a violation. Therefore, the upper bound of the probability of violation is the sum of the total probability of filtered reachable sets and the remaining output sets that violate the property (lines 13 and 25 in Algorithm 3), i.e., $p \leq \bar{p}_v$ for all inputs in the input set Θ. In the best case, all filtered reachable sets do not lead to a violation. Therefore, the lower bound of the probability of violation is the total probability of the remaining output sets that violate the property (line 25 in Algorithm 3), i.e., $p \geq \underline{p}_v$. The probability of an input that is outside of the input set Θ is $1 - \mathcal{P}(\Theta)$. In the worst case, all inputs outside of the input set lead to a violation. This means the maximum probability of the network violating its property for the entire input space is $\bar{p}_v + 1 - \mathcal{P}(\Theta)$. In the best case, all inputs outside of the input set do not lead to a violation. Therefore, the minimum probability of the network violating its property for the entire input space is $p_v^{min} = \underline{p}_v$.

Lemma 8's proof

Proof. The exact scheme uses the exact output reachable sets of the network for verification and conducts the exact counterexample input sets when the network violates its property (lines 19-21). As the counterexample input set shares the same constraints with the violated output set (lines 17 and 20), the total probability of the counterexample input sets equals the total probability of violated output sets.

Rocket lander initial condition

$lb_1(P_1):$	$[-0.2, 0.02, -0.5, -1.0, -20\pi/180, -0.2, 0.0, 0.0, 0.0, -1.0, -15\pi/180]^T$
$ub_1(P_1):$	$[0.2, 0.5, 0.5, 1.0, -6\pi/180, -0.0, 0.0, 0.0, 1.0, 0.0, 0\pi/180]^T$
$lb_2(P_2):$	$[-0.2, 0.02, -0.5, -1.0, 6\pi/180, 0.0, 0.0, 0.0, 0.0, 0.0, 0\pi/180]^T$
$ub_2(P_2):$	$[0.2, 0.5, 0.5, 1.0, -6\pi/180, -0.0, 0.0, 0.0, 1.0, 0.0, 0\pi/180]^T$

Table 4: Initial conditions for the rocket lander benchmark.

N-samples	US-Prob	N-cores	VT
10000	0	1	0.19
10000	0	4	0.16
100000	6e-05	1	1.29
100000	4e-05	4	0.7
10000000	2e-05	1	102.5
10000000	3.94e-05	4	40.30

Table 5: Verification results of ACAXu N_{12} on property 2 using the simple Monte Carlo approach.

ACASXU's full verification results

Prop	Net	p_f	$\mathcal{O}$	$\mathcal{US}-\mathcal{O}$	$\mathcal{C}$	US-Prob-LB	US-Prob-UB	US-Prob-Min	US-Prob-Max	I-Prob	VT
2	1-2	0	45606	25496	25496	1.39251e-05	1.39251e-05	1.39251e-05	0.0134402	0.986574	130.55
2	1-2	1e-05	3190	2397	2397	3.65513e-06	0.0197698	3.65513e-06	0.0331961	0.986574	78.6622
2	1-3	0	114219	53	53	8.91096e-11	8.91096e-11	8.91096e-11	0.0134263	0.986574	416.127
2	1-3	1e-05	4155	0	0	0	0.0397379	0	0.0531642	0.986574	155.064
2	1-4	0	154449	50171	50171	2.16508e-05	2.16508e-05	2.16508e-05	0.0134479	0.986574	518.043
2	1-4	1e-05	4625	2727	2727	1.80812e-07	0.0469075	1.80812e-07	0.0603338	0.986574	180.598
2	1-5	0	122275	121	121	9.42565e-13	9.42565e-13	9.42565e-13	0.0134263	0.986574	348.196
2	1-5	1e-05	3447	0	0	0	0.0282999	0	0.0417262	0.986574	112.726
2	1-6	0	376352	80621	80621	1.87224e-05	1.87224e-05	1.87224e-05	0.013445	0.986574	1424
2	1-6	1e-05	4953	2649	2649	2.80807e-06	0.0528349	2.80807e-06	0.0662612	0.986574	206.772
2	2-1	0	193080	20366	20366	0.0171559	0.0171559	0.0171559	0.0305822	0.986574	636.495
2	2-1	1e-05	4709	224	224	0.00989434	0.0629621	0.00989434	0.0763884	0.986574	183.613
2	2-2	0	471882	320909	320909	0.0353886	0.0353886	0.0353886	0.0488149	0.986574	2102.47
2	2-2	1e-05	6093	4831	4831	0.0195646	0.0940341	0.0195646	0.10746	0.986574	298.999
2	2-3	0	194168	146328	146328	0.0511745	0.0511745	0.0511745	0.0646007	0.986574	707.727
2	2-3	1e-05	4779	4317	4317	0.0371155	0.0941519	0.0371155	0.107578	0.986574	182.177
2	2-4	0	114037	16610	16610	0.0198031	0.0198031	0.0198031	0.0332294	0.986574	362.199
2	2-4	1e-05	4183	257	257	0.0118976	0.047335	0.0118976	0.0607612	0.986574	140.13
2	2-5	0	677346	330546	330546	0.0487406	0.0487406	0.0487406	0.0621669	0.986574	3822.98
2	2-5	1e-05	6009	4840	4840	0.0303233	0.109871	0.0303233	0.123297	0.986574	407.668

2	2-6	0	309365	190910	190910	0.0338494	0.0338494	0.0338494	0.0472757	0.986574	1313.9
2	2-6	1e-05	5475	4367	4367	0.0240279	0.0913298	0.0240279	0.104756	0.986574	249.787
2	2-7	0	679021	335393	335393	0.0622461	0.0622461	0.0622461	0.0756724	0.986574	3532.62
2	2-7	1e-05	6570	4927	4927	0.0405048	0.127336	0.0405048	0.140762	0.986574	452.762
2	2-8	0	585437	283782	283782	0.0427024	0.0427024	0.0427024	0.0561286	0.986574	2723.35
2	2-8	1e-05	5850	4566	4566	0.01958	0.0963161	0.01958	0.109742	0.986574	354.319
2	2-9	0	909914	297873	297873	0.000997678	0.000997678	0.000997678	0.014424	0.986574	4561.2
2	2-9	1e-05	6895	4325	4325	0.000255108	0.106958	0.000255108	0.120384	0.986574	504.504
2	3-1	0	252573	78561	78561	0.0445325	0.0445325	0.0445325	0.0579588	0.986574	1086.38
2	3-1	1e-05	4753	2620	2620	0.0305	0.0726312	0.0305	0.0860575	0.986574	202.726
2	3-2	0	181291	83950	83950	7.21491e-06	7.21491e-06	7.21491e-06	0.0134335	0.986574	727.189
2	3-2	1e-05	4776	3586	3586	2.94061e-20	0.0422196	2.94061e-20	0.0556459	0.986574	174.612
2	3-4	0	133749	64192	64192	0.0175945	0.0175945	0.0175945	0.0310208	0.986574	402.029
2	3-4	1e-05	4286	3088	3088	0.0102766	0.0482358	0.0102766	0.0616621	0.986574	138.466
2	3-5	0	364948	150870	150870	0.0290188	0.0290188	0.0290188	0.0424451	0.986574	1437.35
2	3-5	1e-05	5444	4258	4258	0.017064	0.0816135	0.017064	0.0950397	0.986574	249.336
2	3-6	0	1003429	120329	120329	0.0335763	0.0335763	0.0335763	0.0470026	0.986574	5224.43
2	3-6	1e-05	5485	432	432	0.0207834	0.106866	0.0207834	0.120292	0.986574	451.958
2	3-7	0	475107	54475	54475	0.00404731	0.00404731	0.00404731	0.0174736	0.986574	2598
2	3-7	1e-05	5666	113	113	0.00231884	0.0749959	0.00231884	0.0884222	0.986574	331.1
2	3-8	0	472132	173273	173273	0.0154244	0.0154244	0.0154244	0.0288507	0.986574	2390.2
2	3-8	1e-05	5825	3655	3655	0.00776669	0.083846	0.00776669	0.0972723	0.986574	327.606
2	3-9	0	378803	157622	157622	0.0658148	0.0658148	0.0658148	0.0792411	0.986574	2106.61
2	3-9	1e-05	4269	621	621	0.0286698	0.0812411	0.0286698	0.0946674	0.986574	243.138
2	4-1	0	402334	263892	263892	0.00231247	0.00231247	0.00231247	0.0157388	0.986574	1870.66
2	4-1	1e-05	5771	4743	4743	0.0010368	0.0716188	0.0010368	0.0850451	0.986574	305.28
2	4-3	0	138104	95098	95098	0.0310807	0.0310807	0.0310807	0.044507	0.986574	519.745
2	4-3	1e-05	4379	3426	3426	0.021069	0.0627628	0.021069	0.076189	0.986574	155.109
2	4-4	0	143350	93342	93342	0.0199556	0.0199556	0.0199556	0.0333819	0.986574	519.065
2	4-4	1e-05	4536	3891	3891	0.0136079	0.0491365	0.0136079	0.0625628	0.986574	153.882
2	4-5	0	456909	332642	332642	0.0460214	0.0460214	0.0460214	0.0594477	0.986574	1924.95
2	4-5	1e-05	5269	4834	4834	0.0313368	0.0976121	0.0313368	0.111038	0.986574	285.253
2	4-6	0	1295596	262984	262984	0.0570912	0.0570912	0.0570912	0.0705175	0.986574	7513.59
2	4-6	1e-05	6128	552	552	0.0267682	0.13378	0.0267682	0.147206	0.986574	566.144
2	4-7	0	651996	378735	378735	0.0409502	0.0409502	0.0409502	0.0543764	0.986574	3407.83
2	4-7	1e-05	5940	4476	4476	0.0207855	0.108069	0.0207855	0.121495	0.986574	418.933
2	4-8	0	515006	227035	227035	0.0403188	0.0403188	0.0403188	0.0537451	0.986574	2477.66
2	4-8	1e-05	4909	399	399	0.0118739	0.0727673	0.0118739	0.0861936	0.986574	279.759
2	4-9	0	984139	426879	426879	0.00586402	0.00586402	0.00586402	0.0192903	0.986574	5048.59
2	4-9	1e-05	6388	5526	5526	0.00338763	0.0975845	0.00338763	0.111011	0.986574	517.496
2	5-1	0	201660	93486	93486	0.0277094	0.0277094	0.0277094	0.0411357	0.986574	823.441
2	5-1	1e-05	4733	3510	3510	0.0135803	0.0626989	0.0135803	0.0761252	0.986574	192.558
2	5-2	0	260741	24222	24222	0.0264194	0.0264194	0.0264194	0.0398457	0.986574	940.849
2	5-2	1e-05	4034	332	332	0.0134197	0.061674	0.0134197	0.0751002	0.986574	189.349
2	5-3	0	125749	87	87	1.81747e-09	1.81747e-09	1.81747e-09	0.0134263	0.986574	418.844
2	5-3	1e-05	4130	7	7	1.58834e-09	0.032622	1.58834e-09	0.0460483	0.986574	139.698
2	5-4	0	81317	46659	46659	0.0165051	0.0165051	0.0165051	0.0299314	0.986574	313.873
2	5-4	1e-05	3968	2804	2804	0.0119148	0.0434673	0.0119148	0.0568936	0.986574	124.415
2	5-5	0	212854	178899	178899	0.0458369	0.0458369	0.0458369	0.0592632	0.986574	913.435
2	5-5	1e-05	5120	4668	4668	0.0329328	0.0857782	0.0329328	0.0992045	0.986574	220.873
2	5-6	0	621782	480677	480677	0.0489121	0.0489121	0.0489121	0.0623384	0.986574	3026.36
2	5-6	1e-05	6554	6054	6054	0.0236352	0.116591	0.0236352	0.130017	0.986574	467.549
2	5-7	0	355708	191233	191233	0.066112	0.066112	0.066112	0.0795383	0.986574	1695
2	5-7	1e-05	5162	4115	4115	0.0329694	0.0963858	0.0329694	0.109812	0.986574	252.758
2	5-8	0	544678	441150	441150	0.051441	0.051441	0.051441	0.0648673	0.986574	2696.3
2	5-8	1e-05	6340	6002	6002	0.0272908	0.118407	0.0272908	0.131833	0.986574	391.85
2	5-9	0	619110	344368	344368	0.0531377	0.0531377	0.0531377	0.066564	0.986574	2756.88
2	5-9	1e-05	6130	4801	4801	0.0270714	0.123507	0.0270714	0.136933	0.986574	404.051
3	1-7	0	500	500	500	0.976871	0.976871	0.976871	0.990298	0.986574	3.59073
3	1-7	1e-05	190	190	190	0.980113	0.980432	0.980113	0.993858	0.986574	4.75383
3	1-8	0	393	393	393	0.986574	0.986574	0.987071	1	0.986574	3.2872
3	1-8	1e-05	167	167	167	0.986574	0.986574	0.988565	1	0.986574	4.87869
3	1-9	0	290	290	290	0.980768	0.980768	0.980768	0.994194	0.986574	2.75517
3	1-9	1e-05	61	61	61	0.986574	0.986574	0.993268	1	0.986574	2.60079
4	1-7	0	642	642	642	0.981887	0.981887	0.981887	0.992643	0.989244	3.68531
4	1-7	1e-05	236	236	236	0.989244	0.989244	0.989244	1	0.989244	4.08739
4	1-8	0	397	397	397	0.988684	0.988684	0.988684	0.999439	0.989244	3.00076
4	1-8	1e-05	121	121	121	0.987813	0.988125	0.987813	0.998881	0.989244	3.56073
4	1-9	0	471	471	471	0.989244	0.989244	0.989244	1	0.989244	3.41508
4	1-9	1e-05	142	142	142	0.979634	0.980022	0.979634	0.990777	0.989244	3.6162

Table 6: Full quantitative verification results for all unsafe ACASXu networks. Notations: p_f is the filtering probability, O is the number of output sets, $\mathcal{US}-O$ is the number of unsafe output sets, $US-Prob-LB$ and $US-Prob-US$ are the lower bound and upper bound of the probability of violation, $US-Prob-Min$ and $US-Prob-Max$ are the minimum, and maximum unsafe probability of the network for the entire infinite input space, $I-Prob$ is the input set's probability, VT is the verification time in second.