SAVER: A Toolbox for SAmpling-Based, Probabilistic VERification of Neural Networks*

Vignesh Sivaramakrishnan
vigsiv@unm.edu
University of New Mexico
Albuquerque, New Mexico, USA

Krishna C. Kalagarla
kalagarl@unm.edu
University of New Mexico
Albuquerque, New Mexico, USA

Rosalyn Devonport
devonport@unm.edu
University of New Mexico
Albuquerque, New Mexico, USA

Joshua Pilipovsky
jpilipovsky3@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Panagiotis Tsiotras
tsiotras@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Meeko Oishi
oishi@unm.edu
University of New Mexico
Albuquerque, New Mexico, USA

Abstract

We present a neural network verification toolbox to 1) assess the probability of satisfaction of a constraint, and 2) modify the set to achieve the probability of satisfaction. Specifically, the tool box establishes with a user-specified level of confidence whether the output of the neural network for a given input distribution is likely to be contained within a given set. Should the tool determine that the given set cannot satisfy the likelihood constraint, the tool also implements an approach outlined in this paper to alter the set to ensure that the user-defined satisfaction probability is achieved. The toolbox is comprised of sampling-based approaches which exploit the properties of signed distance function to define set containment.

CCS Concepts

• **Theory of computation → Logic and verification**; • **Mathematics of computing → Probabilistic algorithms**; **Hypothesis testing and confidence interval computation**.

Keywords

Probabilistic Verification, Neural Networks, Stochastic Dynamical Systems, Scenario Optimization, Sample Bounds

ACM Reference Format:
Vignesh Sivaramakrishnan, Krishna C. Kalagarla, Rosalyn Devonport, Joshua Pilipovsky, Panagiotis Tsiotras, and Meeko Oishi. 2025. SAVER: A Toolbox for SAmpling-Based, Probabilistic VERification of Neural Networks. In *28th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '25), May 6–9, 2025, Irvine, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3716863.3718045

*This material is based upon work supported by the National Science Foundation (NSF) under Grant #CNS-1836900, by the Air Force Research Laboratory (AFRL) under Grant #FA9453-23-C-A025, and by National Aeronautics and Space Administration (NASA) under the University Leadership Initiative (ULI) award #80NSSC20M0163. Any opinions, findings, conclusions, and/or recommendations expressed in herein are those of the authors and do not necessarily reflect the views of the NSF, AFRL, or NASA.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

HSCC '25, May 6–9, 2025, Irvine, CA, USA

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-1504-4/2025/05
https://doi.org/10.1145/3716863.3718045

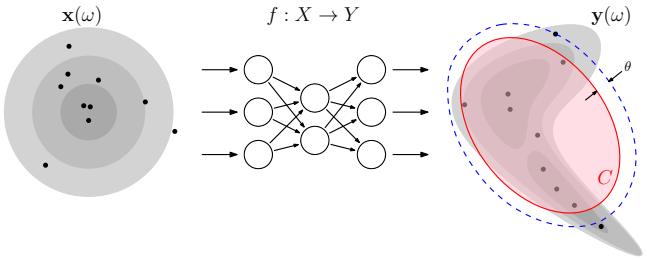


Figure 1: We consider neural nets with probabilistic inputs, and seek to identify the likelihood that the output distribution falls within a set C with at least some probability $1 - \Delta$ (Problem 1), and if necessary, to identify the smallest enlargement of the set C, by a distance θ, in order to satisfy the same probability (Problem 2).

1 Introduction

Neural networks (NN) have shown great promise in myriad domains, often demonstrating performance on part with, or even surpassing, traditional methods. They have even been embedded as elements of safety-critical systems, such as in quadrotor motion planning [1, 2], autonomous driving [3], aircraft collision avoidance [4], and aircraft taxiing [5]. However, the deployment of NNs in safety-critical domains introduces significant challenges, particularly regarding their robustness to input uncertainties and adversarial attacks. These challenges currently represent one of the primary barriers to the broader adoption of NNs in such domains. For instance, in contexts such as malware detection and sparse regression, existing works have shown that adversarial inputs can be crafted with relative ease to exploit vulnerabilities in NNs [6, 7].

To address these concerns, the formal verification of NNs has emerged as a critical area of research. Broadly speaking, verification approaches can be categorized into two types: *deterministic* and *probabilistic*. Deterministic verification provides binary guarantees that the outputs of a NN satisfy a specific condition for all inputs within a defined set. This type of verification is crucial for worst-case safety guarantees. Several state-of-the-art techniques have been developed in this domain, including mixed-integer linear programming [8–11], satisfiability modulo theories [12, 13], semidefinite programming [14–17], and reachability analysis [18].

These methods typically assume a bounded input uncertainty set, centered around some nominal input.

On the other hand, probabilistic verification focuses on assessing the likelihood that a NN satisfies a safety condition under uncertain or random inputs, which may be unbounded. Such uncertainties can arise naturally from environmental noise, signal processing errors, or other exogenous disturbances. Furthermore, adversarial noise, even when imperceptible, can significantly alter NN outputs, particularly in deep networks for tasks like image classification [19, 20].

Probabilistic verification addresses the key question: Given a random input vector $\mathbf{x}_0$ and a neural network f, what is the probability that the output $\mathbf{y} = f(\mathbf{x})$ lies within a predefined safety set C? Specifically, we seek to ensure that this probability exceeds a certain user-defined threshold, expressed as

$$\mathbb{P}_{\mathbf{y}}(C) \geq 1 - \Delta, \tag{1}$$

where $\Delta \in (0, 1)$ represents the satisfaction probability.

A major challenge in probabilistic NN verification is the difficulty of propagating input uncertainty through highly nonlinear mappings. Existing approaches to computing or approximating the chance constraint in (1) can be categorized as either *analytical* or *sampling-based*. Analytical methods include the propagation of confidence ellipsoids using approximate networks, $\tilde{f}$, which reduces the verification problem to a semidefinite program [21], and the use of characteristic functions to propagate input distributions exactly in ReLU networks, drawing on concepts from discrete-time dynamical systems and Fourier transforms [22]. PROVEN [23] approximates the safety probability using linear approximations and concentration inequalities for (sub-)Gaussian inputs with bounded support. Similarly, methods such as CC-Cert [24] combine Cramer-Chernoff bounds with sample propagation to estimate the safety probability, while more recent approaches use branch-and-bound techniques to simultaneously upper and lower bound the probability of safety [25]. Alternatively, scenario-based approaches compute safety certificates by solving chance-constrained programs based on the number of samples, either by directly lower-bounding the success probability [26] or by determining the largest norm ball $\mathbb{B}_\varepsilon$ that is likely to enclose the output [27].

In this paper, we present SAVER: SAmpling-based VERification of Neural Nets, a Python toolbox for sampling-based probabilistic verification. We use the Dvoretzky-Kiefer-Wolfowitz Inequality [28] and scenario optimization approach [29], which provide bounds on the number of samples necessary to validate a specification to a desired probability as in (1). To encode the set specifications, we define the sets as SDFs, whose properties allow us to expand the set should the original specification not achieve the satisfaction probability. We apply this approach on a three examples: 1) the containment of feedforward neural network's output, 2) the robustness of an image classifier, and 3) the resulting position of a aircraft with noise added to the neural network.

2 Problem Formulation

Let $(\Omega, \mathcal{M}(\Omega), P)$ be a probability tuple. The set Ω is the set of all possible outcomes, $\mathcal{M}(\Omega)$ is the set of events, i.e., a σ-algebra, where each event is a set of outcomes, and a function $P : \mathcal{M}(\Omega) \to [0, 1]$ which assigns a probability to each set in the σ-algebra. A random variable is a measurable function, $\mathbf{y} : \Omega \to \mathcal{Y}$. We denote the space of random variables as $\mathbf{y} \in L^p(\Omega, \mathcal{M}(\Omega), P) = \left\{\mathbf{y} : \Omega \to \mathcal{Y} |\ \int_{\mathcal{Y}} |\mathbf{y}(\omega)|^p \, \mathrm{d}P(\omega) < \infty\right\}$. The probability that $\mathbf{y}$ will take on a value in S we represent by a probability measure, $P_{\mathbf{y}}(S) = P(\mathbf{y}^{-1}(S)) = P(\{\omega \in \Omega | \mathbf{y}(\omega) \in S\})$ for $S \in \mathcal{M}(\mathcal{Y})$. We denote a constraint set $C \in \mathcal{M}(\mathcal{Y})$ through a measurable signed distance function (SDF) $g_C : \mathcal{Y} \to \mathbb{R}$,

$$g_C(y) := \begin{cases} \inf_{p \in C} \|p - y\|, & y \in \mathcal{Y} \backslash C, \\ -\inf_{p \in \mathcal{Y} \backslash C} \|p - y\|, & y \in C. \end{cases} \tag{2}$$

Put simply, the SDF says that the point $y \in \mathcal{Y}$ is within the set if its output is negative or zero and positive otherwise. Set operations such as union, intersection, and set difference are readily captured using SDFs. We use SDFs to formulate the two NN verification problems the toolbox solves, stated formally here and depicted graphically in Figure 1. SDFs are ubiquitous in Hamilton-Jacobi-Bellman reachability where one characterizes reachable sets by solving an optimization problem [30].

Problem 1. *Given a neural network, $f : \mathcal{X} \to \mathcal{Y}$, a SDF $g_C : \mathcal{Y} \to \mathbb{R}$, and a probability of satisfaction $\Delta \in (0, 1)$, determine if*

$$\mathbb{P}(\{\omega : g_C(f(\mathbf{x}(\omega))) \leq 0\}) \geq 1 - \Delta, \tag{3}$$

where $1 - \Delta \in (0, 1)$ is the probability of satisfaction.

Problem 2. *Given a neural network, $f : \mathcal{X} \to \mathcal{Y}$, a SDF $g_C : \mathcal{Y} \to \mathbb{R}$, and a probability of satisfaction $\Delta \in (0, 1)$, solve the optimization problem,*

$$\underset{\theta \in \mathbb{R}}{\text{minimize}} \quad \theta, \tag{4a}$$

$$\text{subject to} \quad \mathbb{P}(\{\omega : g_C(f(\mathbf{x}(\omega))) - \theta \leq 0\}) \geq 1 - \Delta. \tag{4b}$$

That is, we find a new set $C^* = \{y \in \mathcal{Y} : g_C(y) - \theta^* \leq 0\}$, where θ^* is the optimal solution of (4), such that it satisfies (1).

Remark 1. *Note that both Problem 1 and 2 are focused on probabilistic verify neural networks. However, this formulation is generic, and so f could also represent a dynamical system with a neural network in the loop, or other functions.*

Solving Problem 1 is crucial for determining whether neural network or a system with a neural network in the loop satisfies specifications with a given probability. We use Problem 2 for:

(1) enlargement: identify how much larger set must be to ensure the satisfaction likelihood $1 - \Delta$
(2) reduction: shrink the set C, in the event that the desired specification can be satisfied with high probability.

3 Sampling-Based Verification

To evaluate the expression on the left hand side of the inequality in (3), we rewrite the probability of neural network output residing in the set as a cumulative distribution function (CDF),

$$\begin{aligned} \Phi_{g_C(f(\mathbf{x}))}(y) &= \mathbb{E}[\mathbf{1}_{\{g_C(f(\mathbf{x})) \leq y\}}], \\ &= \mathbb{P}(\{\omega \in \Omega |\ g_C(f(\mathbf{x}(\omega))) \leq y\}). \end{aligned} \tag{5}$$

We can approximate (5) empirically as,

$$\hat{\Phi}_{g_C(f(\mathbf{x}))}(y) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{\{g_C(f(\mathbf{x}^{(i)})) \leq y\}}. \tag{6}$$

3.1 Dvoretzky-Kiefer-Wolfowitz Inequality

One way to determine how many samples are needed to estimate (5) via (6) is via the Dvoretzky-Kiefer-Wolfowitz (DKW) inequality [28], which provides a bound on the difference between the empirical distribution function $\hat{\Phi}_{g_C(f(\mathbf{x}))}$ and the true CDF $\Phi_{g_C(f(\mathbf{x}))}$. Formally, given N i.i.d. samples, the DKW inequality states that for any error tolerance $\epsilon > 0$,

$$\mathbb{P}\left(\sup_x |\hat{\Phi}_{g_C(f(\mathbf{x}))}(x) - \Phi_{g_C(f(\mathbf{x}))}(x)| > \epsilon\right) \leq 2\mathrm{e}^{-2N\epsilon^2}. \quad (7)$$

Further, to determine the number of samples N an obtain an empirical CDF from (6) that is within ϵ to the actual one in (5) with confidence level $1-\beta$, we can rearrange (7) to obtain,

$$N \geq \left\lceil -\frac{1}{2\epsilon^2} \ln\left(\frac{\beta}{2}\right)\right\rceil. \quad (8)$$

Therefore, solving Problem 1 is merely a matter of using (7) to compute the number of samples via a user specified $\epsilon,\ \beta \in (0,1)$. For Problem 2, we first note that (4) is equivalent to the definition of the quantile function.

Definition 1. *The quantile function $Q : [0,1] \to \mathbb{R}$ is the inverse of the CDF, provided that $\Phi_{g_C(f(\mathbf{x}))}(x)$ is continuous and non-decreasing. Formally, for $p \in [0,1]$, the quantile function is given by*

$$Q(p) = \inf\{x \in \mathbb{R} : \Phi_{g_C(f(\mathbf{x}))}(x) \geq p\}. \quad (9)$$

In other words, $Q(p)$ returns the value x such that the probability of a random variable being less than or equal to x is at least p.

We can even use root-finding, such as the bisection algorithm, to find the smallest θ such that,

$$\hat{\Phi}_{g_C(f(\mathbf{x}))}(\theta^*) = 1 - \Delta. \quad (10)$$

3.2 Scenario Optimization

Scenario optimization constructs data-driven approximations of solutions for the chance-constrained optimization problem

$$\min_{\theta \in \mathbb{R}^{n_\theta}} \quad c^\intercal \theta, \text{ s.t} \quad \mathbb{P}\left(\{\omega : h(\mathbf{x}(\omega), \theta) \leq 0\}\right) \geq 1 - \Delta, \quad (11)$$

where $h : \mathbb{R}^n \times \mathbb{R}^{n_\theta} \to \mathbb{R}$ is convex with respect to its second argument θ. Instead of directly evaluating the chance constraint, which is generally impractical, the scenario approach approximately enforces the constraint in (11) by utilizing samples of $\mathbf{x}$, yielding the scenario relaxation

$$\min_{\theta \in \mathbb{R}^{n_\theta}} \quad c^\intercal \theta, \text{ s.t.} \quad h\left(\mathbf{x}^{(i)}, \theta\right) \leq 0, \quad i = 1, \ldots, N. \quad (12)$$

An optimal solution θ^* of (12) will not be an optimizer of (11) due to the sampling approximation since the set of optimizers is generally of measure zero; however, the set of feasible solutions, i.e., values of θ that satisfy the chance constraint while not necessarily minimizing the objective, has positive measure that we can bound. In particular, a sufficient condition that an optimal solution of (12) be a feasible solution of (11) with probability $\geq 1-\beta$ is that the number of samples N satisfies the bound

$$N \geq \left\lceil \frac{1}{\Delta}\left(\frac{\mathrm{e}}{\mathrm{e}-1}\right)\left(\log\frac{1}{\beta} + n_\theta\right)\right\rceil, \quad (13)$$

where $\beta \in (0,1)$ is the confidence parameter, and n_θ is the number of decision variables. With a user-specified confidence parameter β and the number of decision variables n_θ, we have that the optimal solution θ^* results in satisfaction of the chance constraint with probability $1-\Delta$ with confidence $1-\beta$ [29, 31].

For the neural network verification problem, we can pose the optimization problem in (4) as follows,

$$\min_{\theta \in \mathbb{R}} \quad \theta, \text{ s.t.} \quad g_C\left(f\left(\mathbf{x}^{(i)}\right)\right) - \theta \leq 0, \quad i = 1, \ldots, N, \quad (14)$$

where we can determine the desired number of samples by setting n_θ to 1. Note that, if $\theta^* \leq 0$, then we satisfy the specification, thereby solving Problem 1. Should $\theta^* > 0$, then we address Problem 2, where we determine the expansion of the set to ensure $1-\Delta$ constraint satisfaction.

Note that the optimization problem in (14) is convex, because the decision variable θ is linear. Further, the optimization problem does not require the convexity of the set specified by the SDF. We can solve the optimization problem by Algorithm 1.

Algorithm 1 Scenario-Based Verification Algorithm

Input: Confidence parameter, 1β; probability of satisfaction, Δ; input distribution, $\mathbb{P}_{\mathbf{x}}$; nonlinearity, such as neural network or system, $f : \mathbb{R}^n \to \mathbb{R}^m$; specification, $g_C : \mathbb{R}^m \to \mathbb{R}$

Output: Set modification, θ^*

1: Determine the number samples, N, via (13).
2: Sample N samples of $\mathbf{x}$, i.e. $x^{(i)} \sim \mathbb{P}_{\mathbf{x}}$ for $i = 1, \ldots, N$.
3: Compute $y^{(i)} = g_C\left(f\left(x^{(i)}\right)\right)$ for $i = 1, \ldots, N$.
4: Find the largest $y^{(i)}$ and set $\theta^* = y^{(i)}$.
5: **return** θ^*

3.3 Comparison of Approaches

Both sampling-based approaches prioritize different aspects of the probabilistic nature of probabilistic verification to ensure the probability of satisfaction. The DKW approach enforces probabilistic satisfaction relative to the empirical CDF, which captures the most commonly occurring probability for probabilistic specification satisfaction using the observed samples. In contrast, scenario optimization emphasizes the largest set enlargement such that one satisfies the desired probability of satisfaction from the observed samples.

4 Toolbox Features

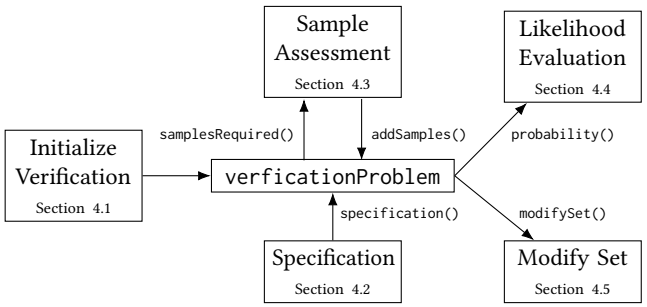


Figure 2: Flowchart depicting SAVER toolbox elements.

The toolbox is downloadable from https://bit.ly/SAVERtoolbox. It runs on Python and requires the following packages: `torch`, `torchvision`, `numpy`, `matplotlib`, `cvxpy`, `scipy` and `jupyterlab`. We implement the toolbox in Python as it contains a number of popular packages for neural network training such as, but not limited to, TensorFlow [32], PyTorch [33], and JAX [34]. By default, mathematical operations are handled by NumPy [35]. Figure 2 overviews the functions within our approach and how one would go about setting up a verification task.

4.1 Initializing a Verification Problem

We can start a sampling-based verification problem by the following functions:

```
# Setup code omitted
verifDKW = verify.usingDKW(betaDKW,epsilonDKW,Delta)
verifScenario = verify.usingScenario(betaScenario,Delta)
```

Here, `verify.usingDKW` and `verify.usingScenario` initialize sampling-based verification problem using DKW or scenario optimization respectively. Both functions take `Delta` corresponding to Δ, the user-defined satisfaction probability. In the function `verify.usingDKW`, the input variables `betaDKW` and `epsilonDKW`, and `Delta` correspond to β and ϵ respectively which appear in (8). In function `verify.usingScenario`, the input variables `betaScenario` corresponds to β in Section 3.2, the confidence for which the probability of satisfaction is greater than $1-\Delta$.

4.2 Adding a Specification

The toolbox consists of two SDFs: 1) norm-ball and 2) polytopes. We present the norm-ball with an example implementation of a two norm-ball (Figure 3),

$$g_C(p) = ||p - c||_2 - r. \tag{15}$$

The function which implements this in our toolbox is:

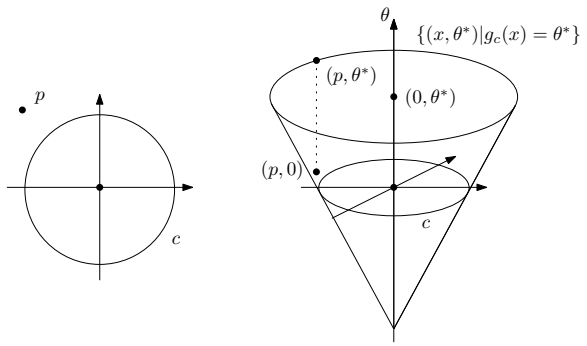


Figure 3: A subset set C (left) of a vector space $\mathcal{X}$ and the graph of its SDF g_C (right). A point p positioned at a signed distance θ^* away from C lies on the border of the enlarged set $C^* = \{x : g_C(x) = \theta^*\}$, which is the projection of the graph section $\{(x, \theta^*) : (g_C(x) = \theta^*)\}$ onto $\mathcal{X}$. Here $\theta^* > 0$, resulting in an enlargement; if θ^* were negative, the new set would be a reduction.

```
# Define SDF:
normSDF = sdfs.norm(center,zeroRadius,norm=2)
```

The function `sdfs.norm` requires variables `center` and `zeroRadius` which correspond to c and r in (15). The function utilizes "`normSDF=`" variable to define what norm the SDF is, and is a two norm by default. For sake of space, we do not expand on the implementation of the polytope SDF except in subsection 5.2 where we utilize it for image robustness. A SDF is added to the verification task calling the following functions with `normSDF` as an input argument:

```
# Add SDF:
verifDKW.specification(normSDF)
verifScenario.specification(normSDF)
```

Should the user desire to change the specification in the verification problem, calling `addSpecification` from the verification problem with a new input variable for the specification, e.g., `newSDF`.

4.3 Adding Samples

Note that for this toolbox, the samples must come from a user-defined sampling function or simulator. To provide the number of samples to the sampler or simulator, we provide the user with the following functions which output the number of samples:

```
# How many samples are needed:
verifDKW.samplesRequired()
verifScenario.samplesRequired()
```

We add samples to verification problem by evaluate the following function with the variable `outputSamples`:

```
# Add samples to the verification problem:
verifDKW.addSamples(outputSamples)
verifScenario.addSamples(outputSamples)
```

Should the user desire to change samples for the verification problem, calling `samples` from the verification problem with a new input variable of samples, e.g. `newOutputSamples`.

4.4 Check Probabilistic Satisfaction of Specification

To address Problem 1, we can call the following function for either sampling-based approach:

```
# Check if the samples satisfy the specification:
verifDKW.probability()
verifScenario.probability()
```

4.5 Modify the Specification to Ensure Probabilistic Satisfaction

Should we not be able probabilistically satisfy the specification, we can solve Problem 2 to determine what how much the set must grow to ensure the satisfaction probability.

```
# Modify the specification:
verifDKW.modifySet()
verifScenario.modifySet()
```

5 Examples

We present three examples that highlight the usage of our sampling-based verification approach. All experiments were run on a Intel 10900K with 128GB RAM running PopOS 22.04 with Python 3.10.12. The examples in Sections 5.1 and 5.2 utilize PyTorch to conduct evaluations of the neural network.

5.1 Feedforward Neural Network with ReLU Activations

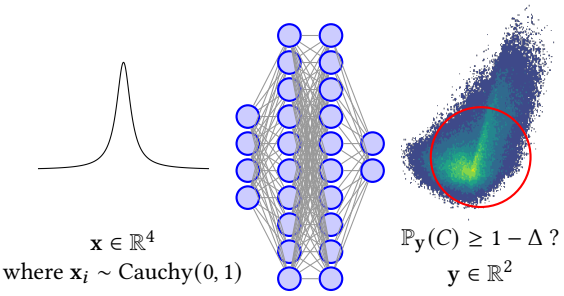


Figure 4: We propagate a standard Cauchy distribution at the input through a feedforward neural network with ReLU activation functions and determine whether the output resides within the set ○, C, with probability greater than $1 - \Delta$.

We employ our sampling-based verification tool on a simple feedforward neural network with ReLU activation functions similar to [22]. As we show in Figure 4, the network takes in samples from a standardized Cauchy distribution, a distribution with undefined moments, passes through the network which consists 4 inputs, 2 layers, each with 10 neurons, and 2 outputs. The set specification we wish to ensure for a satisfaction probability of $1 - \Delta = 0.999$ is a two norm ball of radius 20,000,

$$C = \{y \in \mathbb{R}^2 : ||y||_2 \leq 20,000\}. \tag{16}$$

This results in a SDF,

$$g_C(p) = \|p\|_2 - 20,000. \tag{17}$$

Since a Cauchy distribution is heavy tailed distribution, we set a conservative estimate of how large the set must be, in hopes of reducing the set.

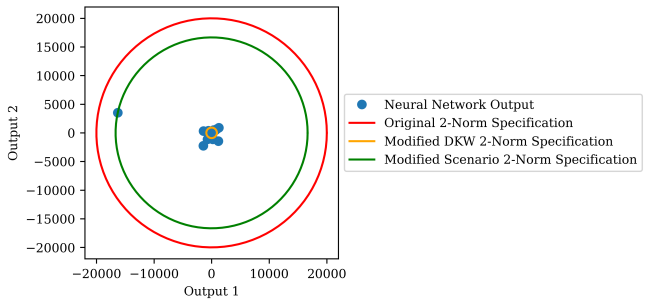


Figure 5: Samples of the output of the neural network are superimposed on the set specification (16). Through both DKW and scenario optimization approaches, we achieve probabilistic satisfaction of the specification with probability greater than 99%. To reduce the set's conservatism, we revise the specification using DKW (yellow) and scenario (green) approaches to solve Problem 2 while ensuring 99.9% specification satisfaction.

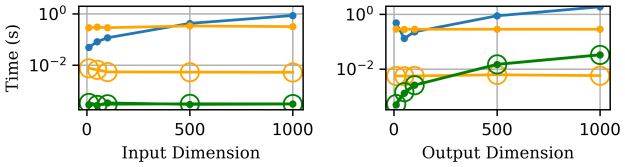


Figure 6: Scaling of sampling ●, verification (DKW ● and Scenario ●), and set modification (DKW ○ and Scenario ○) with respect to input and output dimension.

We run both the DKW and scenario-based approaches from our toolbox and present the results in Figure 5, along with the samples. For the DKW approach, we utilize a confidence, $1 - \beta$, of 0.999 and an error from the true CDF, ϵ, of 0.001, resulting in requiring 3,800,452 samples. The scenario approach uses the same confidence as DKW, requiring 12, 510 samples for the verification task. The analysis we conduct with both the DKW and scenario approaches indicate that we do satisfy the specification with 0.999 probability, thereby addressing Problem 1. Nonetheless, to reduce our initial estimate of the set, we solve (4) in Problem 2 with both the DKW and scenario approach which result in $\theta^*_{\text{DKW}} = -19,232.62$ and $\theta^*_{\text{Scenario}} = -3,779.28$ respectively. Specifically, the DKW-based approach results in a reduction relative the nominal satisfaction probability of the constraint. In contrast, the scenario-based approach results in a smaller set reduction as it prioritizes extreme values to ensure the probabilist of constraint satisfaction.

5.1.1 Analysis of Complexity. Both sample-based approaches theoretically do not depend on the input or output dimension, only the number of samples (Figure 6). Each approach scales differently with increasing samples. The scenario approach is a linear program that scales from polynomial to near-linear with respect to the number of samples, depending on the solver and this is apparent when we scale the output dimension [36, 37]. The DKW approach scales linearly in the number of samples [28] for probabilistic verification and the convergence of the root-finding for set modification converges linearly [38]. Note that for the output scaling, `numpy` runs out of memory beyond a 1000 outputs for 3,800,452 samples.

5.2 Image Classifier

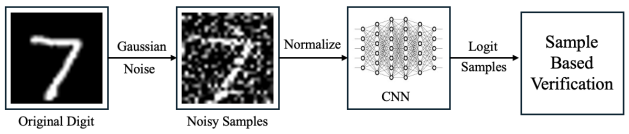


Figure 7: We determine the robustness of our classifier by perturbing each pixel of a fixed digit from the MNIST database with a Gaussian with variance of 0.25, i.e. $\mathcal{N}(0, 0.25)$.

We also validated the sampling-based approach by examining the robustness of a Convolutional Neural Network (CNN) classifier trained on a normalized MNIST dataset [39]. Specifically, we evaluated whether the classifier maintains a high probability of correct

classification on noisy versions of an originally correctly classified image of the digit 7.

To formalize the requirement of correct classification, we define a constraint set C in the logit space $y \in \mathbb{R}^{10}$ of the classifier. Logits are the outputs from the last layer of a neural network before applying the softmax function, representing the raw, unnormalized class scores. The constraint set is a polytope in the logit space which specifies that the classifier's output is '7'. Let $y = f(x)$ denote the logit vector output by the CNN for a normalized input image x, where y_i is the logit corresponding to class i. The condition for the correct classification as '7' is ensured by enforcing that the logit for class '7' (denoted as y_7) is the highest among all classes. In matrix form, these constraints can be expressed as:

$$C = \{y \in \mathbb{R}^{10} : Ay \leq b\}. \tag{18}$$

where each row in A includes a +1 for y_7, a −1 for a particular y_i (where $i \neq 7$), and 0 elsewhere, b is a zero vector.

We then compute the SDF g_C for the polytope C. This function quantifies how far the logit vector y is from the boundary of C, providing a measure of classification stability:

$$g_C(y) = \begin{cases} \min_{z \in C} \|y - z\|_2, & \text{if } y \notin C \\ -\min_i \frac{A_i y - b_i}{\|A_i\|_2}, & \text{if } y \in C \end{cases} \tag{19}$$

where A_i is the i-th row of the matrix A.

- If $y \notin C$, the signed distance is positive and is computed as the shortest Euclidean distance to the polytope's boundary. This distance can be determined by solving the quadratic programming problem outlined above.
- If $y \in C$, the signed distance is negative. It is determined by computing the perpendicular distances from y to each hyperplane defining the polytope's facets. The minimum of these distances represents the shortest distance to the boundary.

We generated input samples by adding standard Gaussian noise with a mean of zero and a variance of 0.25, thus distorting the digit (Figure 7). This pixel-wise noise is added to the un-normalized image, which is then normalized before CNN classification. We require that the specification in (18) be satisfied with $1 - \Delta = 0.99$ probability. To determine if the specification is satisfied via DKW, we specify $\epsilon = 0.01$ and $\beta = 0.001$ for (7), resulting in requiring $38,005$ samples. Likewise, we generated samples needed for the scenario approach with the same confidence as the DKW-based approach, requiring $1,251$ samples. We generated these samples by the above-described process. This resulted in determining that, by Problem 1, we did not satisfy the specification in (18).

5.3 TaxiNet

We validated the sampling-based approach on the TaxiNet benchmark. TaxiNet consists of a neural network which predicts heading angle and the cross track position from images from a camera attached on the right wing of a Cessna 208B Grand Caravan taxiing at 5 m/s down runway 04 of Grant County International Airport [40]. The crosstrack and heading angle are fed into a proportional controller,

$$\phi = -0.74p - 0.44\theta, \tag{20}$$

where ϕ is the steering angle of the aircraft. Gaussian noise to each pixel, i.e. $\mathbf{w} \sim \mathcal{N}(0, 50)$, corrupts the downsampled image that is fed into TaxiNet.

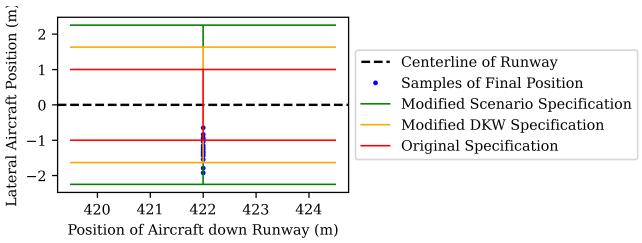


Figure 8: Lateral view of the TaxiNet experiment. The aircraft has an initial cross track deviation of 5 meters, and then taxis 100 meters along the runway. However, solving Problem 1 reveals that because the downsampled images are corrupted, the aircraft cannot regulate its position to within 1 meter of the centerline with at least 90% likelihood. Solving Problem 2 results in enlargements when using either DKW or scenario approaches, with $\theta^* = 0.9$ or 1.25, respectively.

The aircraft starts at cross-track position $p_0 = 5$ meters, heading angle $\theta_0 = 10$ degrees, and runway down-track position of 322 meters. At 422 meters, wish to validate whether the cross-track position is within 1 meter of the centerline of the runway,

$$C = \{p \in \mathbb{R} : |p| \leq 1\}. \tag{21}$$

The SDF is,

$$g_C(p) = |p| - 1, \tag{22}$$

where we have taken the one norm deviation from the aircraft's crosstrack position. Figure 8 visually overviews the experiment and specification, (22), in orange. After we solve the root finding problem that attempts to compute the quantile in (9), solving Problem 2 to find (10).

We require that the specification in (22) be satisfied to with $1 - \Delta = 0.9$ probability. To determine if the specification is satisfied, we specify $\epsilon = 0.1$ and $\beta = 0.001$ for (7), resulting in requiring 381 samples for the DKW-based approach. For the scenario approach, we specify a confidence of $\beta = 0.001$ to (13), requiring 126 samples. We therefore ran the simulator for 381 times over a span of two hours. This resulted in determining that, by Problem 1, we did not satisfy the specification in (22). Thus, we solve Problem 2, to determine a θ that satisfies the probability of satisfaction $1 - \Delta$. Therefore, we proceed to solve (4) in Problem 2 with the DKW and scenario methods, resulting in $\theta^*_{\text{DKW}} = 0.632$ and $\theta^*_{\text{Scenario}} = 1.25$, respectively.

6 Conclusion

We present SAVER, a sampling-based toolbox for probabilistic verification of neural networks. The toolbox provides two approaches to obtain the number of samples necessary to verify whether we satisfy a specification. In addition, the toolbox is also able to modify the set through the usage of SDFs in order to achieve the satisfaction probability. To assess the toolbox's efficacy, we use it on three examples with different objectives. Future work will assess the toolbox against other neural network verification approaches.

References

[1] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural lander: Stable drone landing control using learned dynamics," in *International Conference on Robotics and Automation (ICRA)*, 2019, pp. 9784–9790.

[2] E. Kaufmann, L. Bauersfeld, A. Loquercio, , M. Müller, V. Koltun, and D. Scaramuzza, "Champion-level drone racing using deep reinforcement learning," *Nature*, vol. 620, pp. 982–987, 2023.

[3] Y. Huang and Y. Chen, "Survey of state-of-art autonomous driving technologies with deep learning," in *IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2020, pp. 221–228.

[4] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer, "Policy compression for aircraft collision avoidance systems," in *IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, 2016, pp. 1–10.

[5] D. Cofer, I. Amundson, R. Sattigeri, A. Passi, C. Boggs, E. Smith, L. Gilham, T. Byun, and S. Rayadurgam, "Run-time assurance for learning-based aircraft taxiing," in *AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, 2020, pp. 1–9.

[6] Q. Wang, W. Guo, K. Zhang, A. G. Ororbia, X. Xing, X. Liu, and C. L. Giles, "Adversary resistant deep neural networks with an application to malware detection," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, 2017, p. 1145–1153.

[7] P.-Y. Chen, B. Vinzamuri, and S. Liu, "Is ordered weighted l_1 regularized regression robust to adversarial perturbation? a case study on oscar," in *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2018, pp. 1174–1178.

[8] A. Lomuscio and L. Maganti, "An approach to reachability analysis for feed-forward relu neural networks," 2017.

[9] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. V. Nori, and A. Criminisi, "Measuring neural net robustness with constraints," in *Proceedings of the 30th International Conference on Neural Information Processing Systems (NeurIPS)*. Red Hook, NY, USA: Curran Associates Inc., 2016, p. 2621–2629.

[10] C.-H. Cheng, G. Nührenberg, and H. Ruess, "Maximum resilience of artificial neural networks," in *Automated Technology for Verification and Analysis*. Cham: Springer International Publishing, 2017, pp. 251–268.

[11] V. Tjeng, K. Y. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," in *International Conference on Learning Representations (ICLR)*, 2019.

[12] L. Pulina and A. Tacchella, "Challenging SMT solvers to verify neural networks," *AI Communications*, vol. 25, no. 2, p. 117–135, Apr. 2012.

[13] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *Computer Aided Verification*, R. Majumdar and V. Kunčak, Eds. Cham: Springer International Publishing, 2017, pp. 97–117.

[14] R. A. Brown, E. Schmerling, N. Azizan, and M. Pavone, "A unified view of SDP-based neural network verification through completely positive programming," in *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, vol. 151. PMLR, 28–30 Mar 2022, pp. 9334–9355.

[15] M. Fazlyab, M. Morari, and G. J. Pappas, "Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming," *IEEE Transactions on Automatic Control*, vol. 67, no. 1, pp. 1–15, 2022.

[16] K. D. Dvijotham, R. Stanforth, S. Gowal, C. Qin, S. De, and P. Kohli, "Efficient neural network verification with exactness characterization," in *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, ser. Proceedings of Machine Learning Research, vol. 115. PMLR, 22–25 Jul 2020, pp. 497–507.

[17] S. Dathathri, K. Dvijotham, A. Kurakin, A. Raghunathan, J. Uesato, R. R. Bunel, S. Shankar, J. Steinhardt, I. Goodfellow, P. S. Liang, and P. Kohli, "Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 5318–5331.

[18] J. A. Vincent and M. Schwager, "Reachable polyhedral marching (RPM): A safety verification algorithm for robotic systems with deep neural network components," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021, p. 9029–9035.

[19] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 86–94.

[20] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, pp. 828–841, 2019.

[21] M. Fazlyab, M. Morari, and G. J. Pappas, "Probabilistic verification and reachability analysis of neural networks via semidefinite programming," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 2726–2731.

[22] J. Pilipovsky, V. Sivaramakrishnan, M. Oishi, and P. Tsiotras, "Probabilistic verification of relu neural networks via characteristic functions," in *Proceedings of The 5th Annual Learning for Dynamics and Control Conference*, ser. Proceedings of Machine Learning Research, vol. 211. PMLR, 15–16 Jun 2023, pp. 966–979.

[23] L. Weng, P.-Y. Chen, L. Nguyen, M. Squillante, A. Boopathy, I. Oseledets, and L. Daniel, "PROVEN: Verifying robustness of neural networks with a probabilistic approach," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 09–15 Jun 2019, pp. 6727–6736.

[24] M. Pautov, N. Tursynbek, M. Munkhoeva, N. Muravev, A. Petiushko, and I. Oseledets, "CC-CERT: A probabilistic approach to certify general robustness of neural networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 7975–7983, 06 2022.

[25] D. Boetius, S. Leue, and T. Sutter, "Probabilistic verification of neural networks using branch and bound," 2024.

[26] B. G. Anderson and S. Sojoudi, "Data-driven certification of neural networks with random input noise," *IEEE Transactions on Control of Network Systems*, vol. 10, no. 1, pp. 249–260, 2023.

[27] A. Devonport and M. Arcak, "Estimating reachable sets with scenario optimization," in *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, ser. Proceedings of Machine Learning Research, vol. 120. PMLR, 10–11 Jun 2020, pp. 75–84.

[28] A. Dvoretzky, J. Kiefer, and J. Wolfowitz, "Asymptotic minimax character of the sample distribution function and of the classical multinomial estimator," *The Annals of Mathematical Statistics*, pp. 642–669, 1956.

[29] A. Devonport and M. Arcak, "Estimating reachable sets with scenario optimization," in *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, ser. Proceedings of Machine Learning Research, A. M. Bayen, A. Jadbabaie, G. Pappas, P. A. Parrilo, B. Recht, C. Tomlin, and M. Zeilinger, Eds., vol. 120. PMLR, 10–11 Jun 2020, pp. 75–84.

[30] C. Tomlin, I. Mitchell, A. Bayen, and M. Oishi, "Computational techniques for the verification of hybrid systems," *Proceedings of the IEEE*, vol. 91, no. 7, pp. 986–1001, 2003.

[31] R. Tempo, G. Calafiore, F. Dabbene, *et al.*, *Randomized algorithms for analysis and control of uncertain systems: with applications*. Springer, 2013, vol. 7.

[32] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[33] A. Paszke et al., "PyTorch: an imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[34] J. B. et al., "JAX: composable transformations of python numpy programs," 2018. [Online]. Available: http://github.com/jax-ml/jax

[35] C. R. Harris et al., "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sept. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[36] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[37] Y. Nesterov and A. Nemirovskii, *Interior-point polynomial algorithms in convex programming*. SIAM, 1994.

[38] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical Analysis*, 10th ed. Cengage Learning, 2016.

[39] L. Deng, "The MNIST database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[40] S. M. Katz, A. Corso, S. Chinchali, A. Elhafsi, A. Sharma, M. J. Kochenderfer, and M. Pavone, "NASA ULI aircraft taxi dataset," 2021. [Online]. Available: https://purl.stanford.edu/zz143mb4347