Probabilistic Verification of Neural Networks via Efficient Probabilistic Hull Generation

Jingyang Li[1] **Xin Chen**[2] **Hongfei Fu**[3] **Guoqiang Li**[*1]

[1]Shanghai Jiao Tong University, Shanghai 200240, China, {lijjjjj,li.g}sjtu.edu.cn
[2]University of New Mexico, Albuquerque, NM 87106, USA, chenxin@unm.edu
[3]Shanghai University of Finance and Economics, Shanghai 200433, China, hongfeifu1984@gmail.com

Abstract

The problem of probabilistic verification of a neural network investigates the probability of satisfying the safe constraints in the output space when the input is given by a probability distribution. It is significant to answer this problem when the input is affected by disturbances which are often modeled by probabilistic variables. In the paper, we propose a novel neural network probabilistic verification framework which computes a guaranteed range for the safe probability by efficiently finding safe and unsafe probabilistic hulls. Our approach consists of three main innovations: (1) a state space subdivision strategy using regression trees to produce probabilistic hulls, (2) a boundary-aware sampling method which identifies the safety boundary in the input space using samples that are later used for building regression trees, and (3) iterative refinement with probabilistic prioritization for computing a guaranteed range for the safe probability. The accuracy and efficiency of our approach are evaluated on various benchmarks including ACAS Xu and a rocket lander controller. The result shows a clear improvement over the state of the art.

1 INTRODUCTION

Formal verification of deep neural networks (DNNs) is crucial for ensuring the trustworthiness and the reliability of learning-enabled systems in safety-critical scenarios Meng et al. [2022], Tambon et al. [2022]. While most existing techniques aim at solving the output range analysis problem of DNNs Katz et al. [2017], Wu et al. [2024], Zhang et al. [2018], it is still difficult to know the safety of a DNN when the input is affected by noises that are often modeled by random variables. Since noises or disturbances are often described by Gaussian variables, their values are not bounded and can hardly be handled by existing output range analysis methods. We propose a new approach which generates a guaranteed range for the safety of a DNN when its input is Gaussian variables.

Probabilistic verification Weng et al. [2019], Mangal et al. [2019], Webb et al. [2019], Boetius et al. [2025], Tran et al. [2023] involves analyzing the probability of safety violations under specified input distributions instead of giving a deterministic answer to the safety query. In order to estimate the probability of satisfying the safe constraints in the output space of a DNN, a straightforward approach could be to compute the output distribution and then integrate its density function over the safe output space. However, it is often not feasible due to the complexity of DNNs. In this paper, we propose a method which conservatively estimates the safety of a DNN based on a set of safe or unsafe *probabilistic hulls* that at most intersect on facets. A probabilistic hull is a closed and bounded set in the input space whose orientation is aligned with the Gaussian distribution of the input, such that the probability of it can be efficiently computed based on the error function Bain and Engelhardt [1992]. An upper bound of safety can be obtained by subtracting the probabilities of all probabilistic hulls which are purely unsafe, while a lower bound can be derived by adding the probabilities of all safe hulls. The main contribution of our method is an efficient search strategy for finding large safe and unsafe hulls.

Existing probabilistic neural network verification methods suffer from the curse of dimensionality. Typically, the existing set-based techniques Zhang et al. [2018], Wang et al. [2021] often prioritize partitioning the input space uniformly that cannot recognize critical regions. This leads to exponential amount of subdivisions and is time consuming. Other methods, such as ProbStar Tran et al. [2023] symbolically propagates the input set to the output space and then estimates their safe probability, it is able to more accurately track the input-output dependency however cannot handle the DNNs whose activations are not ReLU.

[*]Corresponding author.

Proceedings of the 42nd Conference on Uncertainty in Artificial Intelligence (UAI 2026), PMLR 337:3785–3801.

We study the probabilistic verification problem on feedforward DNNs. We propose a novel regression tree-guided probabilistic verification framework that achieves both efficiency and generality. Our key insight is that neural network decision/safe boundaries, i.e., the boundary dividing the safe and unsafe set in the output space, can be effectively approximated through adaptive region partitioning guided by sampling-based regression trees, eliminating the unnecessary exploration and subdivision in the regions that are purely safe or unsafe.

Our approach makes three primary contributions: (1) We introduce a regression tree-guided state space splitting strategy for the input space that efficiently generates purely safe or unsafe probabilistic hulls of large probabilities. It avoids the unnecessary subdivisions in the safe or unsafe regions in most practical cases. (2) We propose a boundary-aware sampling method which effectively identifies the safety boundary in the input space by random samples which are later used to build regression trees. (3) We develop an incremental sampling mechanism that iteratively refines the undecided regions with the largest probability, avoiding the prohibitive cost of dense sampling across the entire input space. Experimental evaluation on the challenging benchmarks demonstrates that our method achieves superior accuracy compared to the state of the art and achieves up to a 10× speedup compared to basic branch-and-bound approaches.

2 RELATED WORK

Linear bound propagation techniques, exemplified by CROWN Zhang et al. [2018], have revolutionized neural network verification by extending traditional interval bound propagation to linear function-based bound propagation. This advancement significantly improves scalability while maintaining reasonable tightness for verification tasks. Building upon CROWN, β-CROWN Wang et al. [2021] integrates branch-and-bound (BaB) techniques with GPU acceleration to achieve both completeness and scalability in verification processes.

Probabilistic verification has emerged as a critical paradigm for quantifying neural network safety under stochastic inputs. One part of the research focus on Bayesian neural network verification Cardelli et al. [2019], Wicker et al. [2024a,b]. Early approaches like PROVEN Weng et al. [2019] established distribution-aware robustness guarantees by integrating probabilistic measures with formal verification techniques. Subsequently, hybrid methods Mangal et al. [2019] combined abstract interpretation with Monte Carlo sampling to balance computational efficiency with probabilistic assurances. Statistical evaluation frameworks Webb et al. [2019] further advanced the field by employing adaptive sampling strategies to estimate perturbation tolerance while quantifying violation probabilities. Besides, Pilipovsky et al. [2023] proposes a characteristic-function-based probabilistic verification method which estimates the safe probability via frequency-domain.

Recent developments have focused on scalability. The probabilistic branch-and-bound approach Boetius et al. [2025] iteratively refines probability bounds through dynamic region subdivision, achieving improved verification success rates but suffering from exponential complexity in high-dimensional spaces. Most notably, ProbStar Tran et al. [2023] introduces probabilistic star-based reachability analysis for efficient uncertainty propagation in safety-critical systems. However, ProbStar is limited to ReLU networks due to its reliance on linear region enumeration, restricting its applicability to networks with generic activation functions.

3 PRELIMINARIES

A deterministic input to a DNN can be denoted by a column vector $\mathbf{x} = (x_1, \ldots, x_d)$. Therefore, a noise-affected input can be denoted by $\mathbf{x} + \mathbf{w}$ where $\mathbf{w} = (w_1, \ldots, w_d)$ are the noises for each input variable. In this paper, we focus on Gaussian noises, i.e., $\mathbf{w}$ subject to a multivariate Gaussian distribution, since they are the most popular models for noises in practice. Our framework does not require the input distribution to be Gaussian; it applies to any distribution for which the probability of a compact set in the input space can be efficiently evaluated. Since Gaussian distributions are closed under affine transformations, a noise-affected input can be represented by Gaussian variables, i.e., $\mathbf{x} \sim \mathcal{N}(\mu, \Sigma)$ where μ is the mean and Σ is the covariance matrix.

Problem Formulation. Let $f : \mathbb{R}^d \to \mathbb{R}^m$ denote the input-output mapping of a (feedforward) DNN having d inputs and m outputs. Given a Gaussian input $\mathbf{x} \sim \mathcal{N}(\mu, \Sigma)$ and a safety property in the output space: $\Gamma(\mathbf{y}) : \mathbf{C}\mathbf{y} \geq \mathbf{a}$, such that the property is defined by a finite set of affine constraints/inequalities. Here, $\mathbf{y}$ is a point/vector in the output space, i.e., $\mathbf{y} = f(\mathbf{x})$, $\mathbf{C} \in \mathbb{R}^{k \times m}$ is a matrix of coefficients, and $\mathbf{a} \in \mathbb{R}^k$ is a constant vector. The *DNN probabilistic verification problem* asks to estimate the probability

$$\mathbb{P}(\Gamma(\mathbf{y}) \,|\, \mathbf{x} \sim \mathcal{N}(\mu, \Sigma)). \tag{1}$$

As we pointed out that an exact calculation of the probability (1) is intractable. Therefore, we conservatively estimate an upper bound as well as a lower bound for the actual probability via computing a finite set of safe and unsafe probabilistic hulls.

Given a set of Gaussian variables $\mathbf{x} \sim \mathcal{N}(\mu, \Sigma)$ such that $\Sigma = diag(\sigma_1^2, \ldots, \sigma_d^2)$, i.e., all variables subject to independent distributions. The probability $\mathbb{P}(\mathbf{x} \in H)$, or simply $\mathbb{P}(H)$, where the hull H is a box $[a_1, b_1] \times \cdots \times [a_d, b_d]$ can be computed by evaluating the expression $\prod_{1 \leq i \leq d} \mathbb{P}(a_i \leq$

$x_i \leq b_i$) such that

$$\mathbb{P}(a_i \leq x_i \leq b_i) = \frac{1}{2}\left(\text{erf}(\frac{b_i-\mu}{\sqrt{2}\sigma_i}) - \text{erf}(\frac{a_i-\mu}{\sqrt{2}\sigma_i})\right)$$

where erf is the error function Bain and Engelhardt [1992]. For any full rank covariance matrix Σ, a hull H aligned with its orientation can be calculated in a similar way. That is, we may transform Σ to a diagonal matrix, transform H to a box using the same mapping and evaluate the probability of being in H with the transformed distribution. Hence, for simple denotation, we assume that Σ is diagonal. In a DNN verification problem, a hull H is safe (unsafe resp.) when all points in its output set $f(H)$ satisfying (violating resp.) the safe property Γ.

4 MAIN APPROACH

The main idea of our approach for estimating conservative upper and lower bounds for the actual safe probability is to iteratively construct a set of safe hulls: $H_1^s, \ldots, H_N^s$ and a set of unsafe hulls: $H_1^u, \ldots, H_M^u$, such that all hulls intersect at most on the facets. Therefore, each hull represents an independent probability from the others. Then, the upper bound is computed as $U_s = 1 - \sum_{1\leq i\leq M} \mathbb{P}(H_i^u)$, while the lower bound is obtained as $L_s = \sum_{1\leq i\leq N} \mathbb{P}(H_i^s)$.

Theorem 1. *If $H_1^s, \ldots, H_N^s$ are safe and $H_1^u, \ldots, H_M^u$ are unsafe, then we have that $U_s \geq L_s$, and the actual safe probability must be in the interval $[L_s, U_s]$.*

Obviously, if the probabilistic hulls can be efficiently found and have high probabilities, an accurate range for safe probability can be obtained. Unlike the related methods, the probabilistic hulls are not generated from random samples. Instead, we build (safe) boundary-ware regression trees which subdivide the input space into box regions, then we use CROWN to verify the safety of them and identify the safe and unsafe subregions as hulls. In the experiments, we show that such a strategy greatly outperforms the method using uniform sampling.

The high-level workflow of our framework is given in Figure 1. Given a DNN f, a safety property Γ, a Gaussian distribution $\mathcal{N}(\mu, \Sigma)$ for the input, and a termination threshold $\varepsilon > 0$. Here, we only need to use the DNN as a blackbox. The main algorithm keeps three sets of probabilistic hulls: $R_{\text{safe}}, R_{\text{unsafe}}, R_{\text{unknown}}$ that are used to keep the set of safe, unsafe and unknown hulls respectively. Initially, R_{unknown} has a single element which represents the whole state space, while R_{safe}, R_{unsafe} are both empty. The whole state space can be represented by a large bounded set which is of a probability sufficiently close to 1 (e.g. $\geq 99.999\%$) for the inputs, instead of an unbounded support range from Gaussian distributions. Then the main loop performs the following steps: (1) It chooses the hull R_{next} which has the highest probability in R_{unknown} and delete it from R_{unknown}; (2) R_{next} is subdivided using our boundary-aware subdivision strategy which builds a regression tree and the subdivisions are new probabilistic hulls obtained as the tree leafs; (3) We verify the output sets of the subdivisions. We add the safe hulls to R_{safe}, the unsafe hulls to R_{unsafe}, and the hulls that are not purely safe or unsafe to R_{unknown}. The loop terminates when the summation of the probabilities of the hulls in R_{unknown} is $< \varepsilon$. The details are described in the following subsections.

4.1 BOUNDARY-AWARE SUBDIVISION STRATEGY

Efficiently finding large safe/unsafe hulls plays a key role in our framework. A naive way could be uniformly subdivide the input state space and verify the safety of each piece. However, it is often very expensive and ignores the decision boundary. Directly considering the boundary in probabilistic hull search can be difficult, since the boundary is in the output space and the subdivision in performed in the input space. However, it is still possible to "backpropagate" the information of the boundary to the subdivision process. Figure 2 gives an intuitive comparison between uniform subdivision and boundary-aware subdivision. The red curve indicates the backpropagated boundary to the input space, it is often not computable and unknown to our process. It can be seen from the figure that, using boundary-aware subdivision can better produce large hulls which do not intersect the boundary and are pure safe or unsafe.

Our boundary-aware subdivision method is outlined in Algorithm 1. Given a DNN f, a region R in the input space, number of samples n and the safety specification defined by the parameters $\mathbf{C}$ and $\mathbf{a}$. Our whole *boundary-ware subdivision* process performs the following steps: (1) A set S_{base} of n points is sampled within the given region R, and the safety of their outputs is checked; (2) If both safe and unsafe outputs are found, it conducts boundary-aware sampling by rejecting the samples whose outputs are sufficiently far from the boundary, and we obtain a point set S; (3) Otherwise, we set $S = S_{\text{base}}$ since the boundary might not be in R; (4) We use the samples in S to build a regression tree T in the region R, where each leaf represents a subdivision of R. The details of boundary-aware sampling are given as below.

Combined Sampling Strategy Foundation. We introduce the setup of our base sampling strategy. Our base sampling strategy is the combination of the distributional sampling of the input distribution $\mathcal{N}(\mu, \Sigma)$ and the uniform distribution. Given an integer n, we sample $\lfloor n/2 \rfloor$ points with distributional sampling and sample $n - \lfloor n/2 \rfloor$ points with uniform sampling. We adopt the combination instead of only one of them because (i) we need to take the distribution of the input variables into account since it helps to find probabilistic hulls in the space of high probability density and make the hulls of high probability, while (ii)

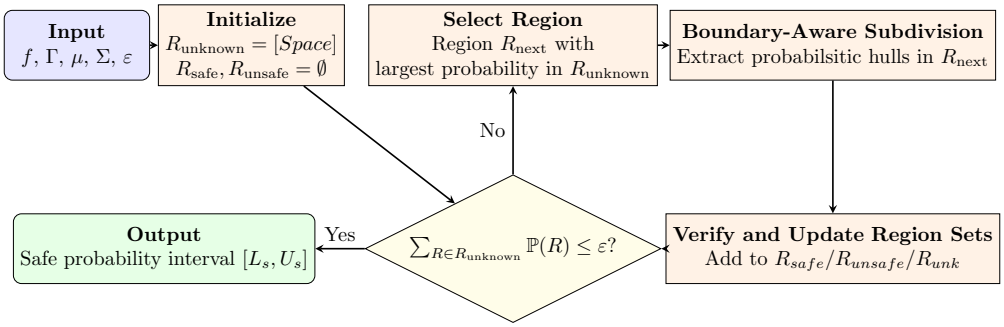

Figure 1: Workflow of our regression tree-guided probabilistic verification framework.

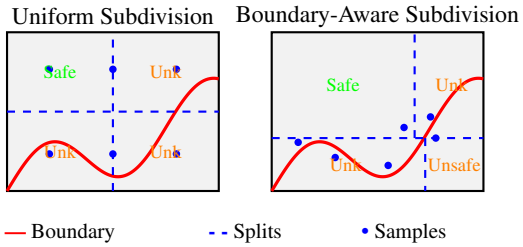

Figure 2: Uniform vs. boundary-aware subdivision.

additionally considering a uniform distribution helps us to sample the margin of a region and have balanced safe and unsafe samples.

Boundary Awareness Through Elimination. In order to better reflect the safe boundary in R, we check our samples to see if both safe and unsafe points are generated. If not, we do not start the rejection sampling step. Otherwise, we aim at keeping the samples near the boundary. Although it is difficult to compute the boundary in the input space, the distance of a sample $\mathbf{x}$ to the boundary can be measured as the distance between $f(\mathbf{x})$ and the boundary. Then we apply an elimination mechanism to remove the samples that are sufficiently far from the boundary. To do so, we associate a probability distribution for eliminating a sample. For each sample $\mathbf{x}$, we compute its output $f(\mathbf{x})$ and the distance to the safety boundary $\mathbf{C}f(\mathbf{x}) - \mathbf{a}$. Then, the elimination probability is given by

$$\mathbb{P}_e(\mathbf{x}) = 1 - \exp(-\text{rank}(|\mathbf{C}f(\mathbf{x}) - \mathbf{a}|/\sigma)) \qquad (2)$$

where rank returns the corresponding index in the sorted distance from the nearest to the farthest.

Regression Tree Construction. We use the sample set after elimination to build a regression tree which is expected to better split the region along the boundary. The tree construction follows the standard algorithm Mitchell [1997]. For each non-leaf node, the impurity of the samples is measured as

$$\text{Impurity} = \frac{\text{MSE}(S_{\text{left}}, S_{\text{right}})}{L^{\alpha}}$$

where $S_{\text{left}}, S_{\text{right}}$ are the split sets, L is the variation of the samples in the dimension of subdivision, and α is a user-specified hyperparameter controlling the geometric penalty. The MSE is computed as $\text{MSE}(S_{\text{left}}, S_{\text{right}}) = \text{Var}(y_i | x_i \in S_{\text{left}}) \cdot |S_{\text{left}}| + \text{Var}(y_i | x_i \in S_{\text{right}}) \cdot |S_{\text{right}}|$. Each subdivision is performed to minimize the impurity of the two subdivisions.

Algorithm 2 presents the main algorithm that is a detailed treatment of the workflow in Figure 1. It operates through an iterative refinement loop that progressively subdivides the input space. Rather than attempting to resolve all unknown hulls equally, we strategically select and refine only the candidates with the highest probability mass among the unknown hulls.

4.2 ANALYSIS OF THE FRAMEWORK

Correctness of the Result. We investigate the correctness of the result from Algorithm 2. First, all of the probabilistic hulls generated intersect at most on the facets. To see that, each hull is obtained as a leaf of a regression tree which subdivide a rougher hull into sub-hulls whose intersections are at most the facets. Therefore, each probabilistic hull in the set R_{safe} and R_{unsafe} has a probability which is independent from the probabilities represented by the other hulls. Next, we show that U_s is an over-estimate of the actual safe probability and L_s is an under-estimate of probability. Due to the conservativeness of the tool CROWN, all hulls in R_{unsafe} are unsafe and all hulls in $\sum_{R \in R_{\text{safe}}}$ are safe. Since the probabilities of the hulls in R_{unsafe} are independent, we infer that the DNN is at least $1 - U_s = \sum_{R \in R_{\text{unsafe}}} \mathbb{P}(R)$ unsafe, and thereby at most U_s safe. Analogously, the DNN is at least $L_s = \sum_{R \in R_{\text{safe}}} \mathbb{P}(R)$ safe since the hulls are safe and represent an independent probability. We also have that $L_s \leq U_s$.

Termination of the Main Algorithm. We show that Algorithm 2 will eventually terminate with $U_s - L_s < \varepsilon$. Our

Algorithm 1: Boundary-Aware Subdivision

Input: Region R, number of samples n, network f, $\mathbf{C}$, $\mathbf{a}$

Output: A regression tree T that partitions R along the safety boundary

/* Step 1: Combined sampling -- mixture of $\mathcal{N}(\mu, \Sigma)$ and uniform */

$S_{\text{base}} \leftarrow \text{COMBINEDSAMPLE}(R, n)$;

/* Step 2: If outputs are all safe or all unsafe, the boundary is unlikely to lie in R; build the tree directly */

if *S_{base} contains only safe or unsafe points* **then**

  return S_{base}

end

/* Step 3: Boundary-aware rejection -- discard points far from the boundary using elimination probability $\mathbb{P}_e$ */

$S \leftarrow \emptyset$;

$n_{\text{attempts}} \leftarrow 0$;

while $|S| < n$ ***and*** $n_{attempts} < max\_attempts$ **do**

  $\mathbf{x} \leftarrow \text{COMBINEDSAMPLE}(R, n)$ $\mathbf{x}_{\text{accept}} \leftarrow$ $\text{ELIMINATESAMPLES}(\mathbf{C}, f, \mathbf{x})$; // Eq. (2)

  $S \leftarrow S \cup \{\mathbf{x}_{\text{accept}}\}$;

  $n_{\text{attempts}} \leftarrow n_{\text{attempts}} + 1$;

end

/* Step 4: Top up with unrestricted samples if rejection produced too few */

if $|S| < n$ **then**

  $S \leftarrow S \cup \text{COMBINEDSAMPLE}(R, n - |S|)$;

end

/* Step 5: Build the tree on the boundary-aware sample set */

$T \leftarrow \text{BUILDREGRESSIONTREE}(S, R)$

return T

Algorithm 2: The Main Algorithm

Input: Network f, $\mathbf{C}$, $\mathbf{a}$, distribution $p(\mathbf{x})$, threshold ε

Output: Safe probability interval $[L_s, U_s]$

Initialize: $R_{\text{unknown}} \leftarrow$ [entire_input_space];

$R_{\text{safe}} \leftarrow \emptyset$, $R_{\text{unsafe}} \leftarrow \emptyset$;

while $\sum_{R \in R_{unknown}} \mathbb{P}(R) \geq \varepsilon$ **do**

  // Select region with maximum probability mass

  $R_{\text{next}} \leftarrow$ SelectLargestProbRegion(R_{unknown});

  // Targeted sampling within selected region

  $S \leftarrow$ BoundaryAwareSample(R_{next}, small_sample_size);

  // Build regression tree and extract leaf regions

  $T \leftarrow$ BuildRegressionTree(S, R_{next});

  $\{R_1, ..., R_k\} \leftarrow$ ExtractLeafRegions(T);

  // Verify each leaf region

  $\{s_1, ..., s_k\} \leftarrow$ CROWNVerify($\{R_1, ..., R_k\}$, $\mathbf{C}$, $\mathbf{a}$);

  // Update region sets

  Remove R_{next} from R_{unknown};

  for $i \leftarrow 1$ **to** k **do**

    if $s_i =$ *Safe* **then**

      $R_{\text{safe}} \leftarrow R_{\text{safe}} \cup \{R_i\}$;

    else if $s_i =$ *Unsafe* **then**

      $R_{\text{unsafe}} \leftarrow R_{\text{unsafe}} \cup \{R_i\}$;

    else

      $R_{\text{unknown}} \leftarrow R_{\text{unknown}} \cup \{R_i\}$;

    end

  end

end

$L_s \leftarrow \sum_{R \in R_{\text{safe}}} \mathbb{P}(R)$;

$U_s \leftarrow 1 - \sum_{R \in R_{\text{unsafe}}} \mathbb{P}(R)$;

return $[L_s, U_s]$

main algorithm can be viewed as a new branch-and-bound scheme with probability density and boundary-aware subdivision. During the iterations, the sum of the volumes of the hulls in the unknown set R_{unknown} becomes smaller and smaller due to the repeated refinement. A proof for the termination can be directly obtained by adapting the termination proof of the standard branch-and-bound method Jaulin et al. [2001]. The volume of the region defined by the union of all undecided hulls, i.e., the hulls in the set of R_{unknown}, converges to zero when the number of iterations goes to infinity. Hence, the total probability of all hulls in R_{unknown} will eventually be below the given threshold ε after a finite number of iterations. It is also the probability that lies out of the confirmed safe and unsafe probability estimates, i.e., $U_s - L_s < \varepsilon$.

Theorem 2. *Algorithm 2 terminates with an interval that contains the actual safe probability of the DNN.*

When Algorithm 2 terminates, the union of the hulls in R_{unknown} forms an overapproximation of the image of the safety boundary via the inverse mapping of the DNN.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Platform and Tools. Our experiment is conducted on a Linux server running Ubuntu 18.04, equipped with an Intel Xeon E5-2678 v3 processor and a GPU of Nvidia RTX 4090. We use auto_LiRPA Xu et al. [2020] to compute the bounds, and our implementation takes as input neural networks in PyTorch format. The safety verification of

probabilistic hulls are performed by CROWN Zhang et al. [2018]. For all benchmarks, each input dimension follows an independent Gaussian $\mathcal{N}(\mu_i, \sigma_i^2)$ with μ_i at the centre of the spec range and $\sigma_i = (\text{ub}_i - \mu_i)/3$, so that the spec range covers $\mu_i \pm 3\sigma_i$ in each dimension. For our method and BaB, the CROWN bounds are computed with GPU in parallel mode and with CPU in serial mode. ProbStar uses its own Star-set propagation on CPU, which does not share the same backend.

Our full source code, scripts, and necessary experimental data are publicly available on GitHub at `https://github.com/LIJYann/Regression-Tree-Guided-Probabilistic-Hull-Generation`. The repository corresponds to the code version used for this submission, ensuring reproducibility.

Comparison Baselines. We compare our approach with two state-of-the-art tools, ProbStar Tran et al. [2023] and the basic branch-and-bound (BaB) method Boetius et al. [2025] (binary partition on the longest dimension). The comparison is done on three groups of DNN benchmarks: ACAS Xu, the rocket lander controller benchmark for SpaceEx Falcon9, and the tanh-based DNNs distilled from those in ACAS Xu (see Gou et al. [2021]). We use a less restrictive condition, $\max_{R \in R_{\text{unknown}}} \mathbb{P}(R) \leq \varepsilon$, for the termination condition in order to reach a reasonable tradeoff between efficiency and accuracy. This condition is used in both our method and the BaB in order to give a fair comparison. The parallel computation in BaB and our approach are conducted on GPU with auto_LiRPA. However, due to the implementation of ProbStar, it can only be parallelized on CPU, and we use the default parallel setting for ProbStar. The experimental results from the tools are compared based on the runtime and the tightness of the upper and lower bounds for the actual safe probability.

Hyperparameter Settings. We perform a grid search over three groups of hyperparameters: (i) sampling weights $(w_{\text{uniform}}, w_{\text{distribution}})$, (ii) regression-tree depth, and (iii) impurity-scheduler parameters (α, β). The impurity scheduler combines the α-weighted impurity measure with longest-dimension splitting: initial splits follow the longest dimension until the hull probability reaches the threshold β, after which it switches to the α-weighted measure. Full details are in Appendix A.

Illustrative Toy Examples. Before we show the comparisons, Figure 3 illustrates the probabilistic hulls computed for the toy example described in Tran et al. [2023]. The red boxes are the probabilistic hulls that are identified as unsafe. The green boxes are the safe hulls. While the yellow ones are the boxes whose corresponding outputs intersect the safety boundary. The union of the yellow boxes forms an overapproximation of the image of the safety boundary under the inverse mapping of the neural network. It can be seen that the state far from the boundary is often included by large probabilistic hulls, i.e., some unnecessary subdivisions can be avoided using our boundary-aware method. The rest of the section addresses the experimental evaluation goals as mentioned previously.

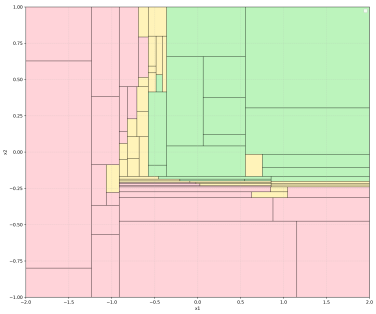

Figure 3: Boundary-aware subdivision for the toy example.

5.2 ACAS XU BENCHMARK

The ACAS Xu collision avoidance system benchmarks Manfredi and Jestin [2016] consist of 45 networks (5×9 configurations) with 5 inputs, 6 layers, and 50 neurons per layer. We test Property P2 in the benchmarks which requires the network not to output "COC" as the optimal action. This means that the network output is **unsafe** when the first output is the minimal. In this comparison, all methods filter regions (branches in ProbStar's case) smaller than 10^{-5}. We use 1000 initial sampling points and 100 iterative sampling points in our approach. The hyperparameter of this benchmark is pure distributional sampling (0.0,1.0), depth 5, $\beta = 0.75$, $\alpha = 0.05$.

Table 1 shows the results for all tools, where the best bounds and runtimes are marked in bold (and for all subsequent tables). From the table, we observe that BaB times out on all the benchmarks, and our method mostly generates better bounds than ProbStar. Specifically, our bounds uniformly outperforms ProbStar in the bounds for unknown probabilities $U_s - L_s$, and our runtime is comparable with ProbStar. Besides, the parallel paradigm originated from CROWN yields substantial performance gains, with an average speedup of 4.8× compared to our non-parallel version.

5.3 ROCKET LANDER CONTROLLER BENCHMARK

To further demonstrate the ability of our method to handle high-dimensional neural networks, we conduct evaluation on the Rocket Lander Yang et al. [2022], which simulates the vertical landing control of a SpaceX Falcon 9 first-stage rocket. The objective is to safely land the rocket on a sea-based platform from a given height, which requires precise control of both the rocket's velocity and lateral angle.

Table 1: Results on ACAS Xu Benchmark for Property P2. Legend - **Net**: DNN benchmarks, **Approach**: our approach ("Ours"), ProbStar and BaB, L_s: the lower bound of the safety probability interval, U_s: the upper bound of the safety probability interval, $U_s - L_s$: upper bound for the unknown probability, Time (non parallel): the runtime without parallel computing in seconds, Time (parallel): the runtime with parallel computing in seconds, T.O.: Time out, >2 hours.

Net	Approach	L_s	U_s	$U_s - L_s$	Time (non parallel)	Time (parallel)
1-6	Ours	**0.984714**	1.000000	**0.015286**	**124.21**	**46.58**
1-6	ProbStar	0.933739	**0.997192**	0.063453	2840.35	315.75
1-6	BaB	-	-	-	T.O.	T.O.
2-2	Ours	**0.924122**	0.991622	**0.067500**	**1248.70**	**240.79**
2-2	ProbStar	0.892540	**0.980435**	0.087895	3869.90	430.20
2-2	BaB	-	-	-	T.O.	T.O.
2-9	Ours	**0.954046**	0.999941	**0.045895**	**819.22**	**249.24**
2-9	ProbStar	0.879616	**0.999745**	0.120129	6715.75	746.56
2-9	BaB	-	-	-	T.O.	T.O.
3-1	Ours	**0.923320**	0.972056	**0.048735**	**949.94**	**189.76**
3-1	ProbStar	0.913943	**0.969500**	0.055557	2462.36	273.73
3-1	BaB	-	-	-	T.O.	T.O.
3-6	Ours	**0.914964**	0.982248	**0.067284**	**1808.11**	**359.85**
3-6	ProbStar	0.879708	**0.979217**	0.099509	5191.90	577.16
3-6	BaB	-	-	-	T.O.	T.O.
3-7	Ours	**0.923570**	0.999888	**0.076318**	**1817.89**	**386.33**
3-7	ProbStar	0.911578	**0.997681**	0.086103	4187.35	465.49
3-7	BaB	-	-	-	T.O.	T.O.
4-1	Ours	**0.961700**	0.999953	**0.038253**	**884.45**	**155.31**
4-1	ProbStar	0.914955	**0.998963**	0.084008	3120.21	346.86
4-1	BaB	-	-	-	T.O.	T.O.
4-7	Ours	**0.913432**	0.979347	**0.065914**	**1604.20**	**381.73**
4-7	ProbStar	0.878505	**0.979215**	0.100710	3917.94	435.54
4-7	BaB	-	-	-	T.O.	T.O.
5-3	Ours	**0.973373**	**1.000000**	**0.026627**	**336.51**	**81.21**
5-3	ProbStar	0.953952	**1.000000**	0.046048	1306.79	145.27
5-3	BaB	-	-	-	T.O.	T.O.

The control system outputs three components: a main engine thruster ($F_E \in [0, 1]$) with an adjustable angle (ϕ), and two side nitrogen thrusters ($F_S \in [-1, 1]$, where -1 and 1 indicate full throttle of right and left thruster respectively). The system state is represented by a 9-dimensional vector: $(x, y, v_x, v_y, \theta, \omega, F'_E, F'_S, \phi')$, where (x, y) is the position relative to the landing center; (v_x, v_y) are the horizontal and vertical velocities; θ is the lateral angle; ω is the angular velocity; and (F'_E, F'_S, ϕ') are the previous control actions.

We directly use the control policy network provided in ProbStar. This network consists of 9 inputs, 3 outputs, and 5 hidden layers each containing 20 ReLU neurons. We focus on two critical safety properties that ensure the effectiveness of tilt control: P_1 (Right Tilt Prevention) specifies that for system states where $\theta \in [-20^\circ, -6^\circ]$, $\omega < 0$, $\phi' \leq 0$, and $F'_S \leq 0$, the network output must satisfy either $\phi < 0$ or $F_S < 0$; P_2 (Left Tilt Prevention) specifies that for system states where $\theta \in [6^\circ, 20^\circ]$, $\omega \geq 0$, $\phi' \geq 0$, and $F'_S \geq 0$, the network output must satisfy either $\phi > 0$ or $F_S > 0$.

We conduct experiments on the Rocket Lander controller to compare our method with ProbStar. The hyperparameter is mixed sampling ($w_{\text{uniform}} = 0.25, w_{\text{distribution}} = 0.75$), depth 5, $\beta = 0.0$, $\alpha = 0.05$. In this comparison, all methods filter regions (branches in ProbStar's case) smaller than 10^{-3}. We do not run BaB since it cannot handle high-dimensional input due to inherent combinatorial explosion. For our method, we use 9000 initial sampling points and 900 iterative sampling points to adapt to the high input dimension of this benchmark. Similar to the setting in Table 1, we compare both the bounds and the runtime (including parallel and non-parallel). The results are shown in Table 2 (non-parallel) and Table 3 (parallel). From these tables, we observe that our method consistently generates significantly tighter bounds for the unknown probability ($U_s - L_s$) in all configurations. Furthermore, our method demonstrates superior time efficiency, achieving substantial runtime reductions in both non-parallel and parallel settings.

Table 2: Experiments on Rocket Lander Controller (non-parallel). **Net** & **Prop** means different configurations in the original benchmark. Others are the same as Table 1.

Net & **Prop**	**Approach**	$U_s - L_s$	**Time (s)**
0 & 1	ProbStar	0.715046	418.183
0 & 1	Ours	**0.650548**	**171.933**
0 & 2	ProbStar	0.831752	597.775
0 & 2	Ours	**0.262993**	**85.482**
1 & 1	ProbStar	0.554032	438.278
1 & 1	Ours	**0.229874**	**70.723**
1 & 2	ProbStar	0.356913	319.571
1 & 2	Ours	**0.079113**	**23.287**

Table 3: Experiments on Rocket Lander Controller (parallel). The legend is the same as Table 2.

Net & **Prop**	**Approach**	$U_s - L_s$	**Time (s)**
0 & 1	ProbStar	0.715046	141.047
0 & 1	Ours	**0.651642**	**33.767**
0 & 2	ProbStar	0.831752	225.331
0 & 2	Ours	**0.265237**	**23.573**
1 & 1	ProbStar	0.554032	225.331
1 & 1	Ours	**0.230962**	**18.900**
1 & 2	ProbStar	0.356913	117.494
1 & 2	Ours	**0.079063**	**12.947**

5.4 EFFICIENCY OF FINDING PROBABILISTIC HULLS

We compare the efficiency of our method and the BaB on finding safe and unsafe probabilistic hulls, since both of the methods use probabilistic hulls to estimate the safe probability. We use the distilled DNNs Gou et al. [2021] of the ones in the ACAS Xu set. All of the new DNNs have $\tanh$ activation function. Moreover, we consider the parallel setting for both our method and BaB. The hyperparameters are the same as in the original ACAS Xu experiments.

The comparison is made in the following way. We set up an end time of 1800 seconds to let both of the methods find probabilistic hulls. Each of the methods may terminate either at the end time or when the early termination condition holds (i.e., the highest probability of the undetermined sets is less than the given threshold ε, similar to $\max_{R \in R_{\text{unknown}}} \mathbb{P}(R) \leq \varepsilon$). Then we compare the runtime and the estimated upper and lower bounds for the safe probability. Clearly, the tighter the bounds, the better the result.

We use $\varepsilon = 10^{-5}$, and the experimental results are given in Table 4. It can be seen that our method always stops before the end time and the runtime are much lower than the basic BaB method. On the other hand, the bounds obtained from our method are also much tighter than those from the basic BaB ones. Hence, our method is much more efficient in finding safe and unsafe probabilistic hulls.

5.5 DISCUSSION

Basically, our method can handle most of the feedforward DNNs with various activation functions, since we use the given DNN as a blackbox to produce the samples. This is a clear advantage over the state-of-the-art tools such as ProbStar which requires neural networks only to have ReLU activation functions. On the other hand, our boundary-aware subdivision demonstrate a great performance improvement in not only time efficiency but also accuracy comparing to the basic branch-and-bound method.

In the experiments, we also find two weaknesses of our method. First, the total runtime highly depends on the time spent by CROWN which is used to verify the safety of a hull in the input space. It can be improved by designing a more efficient output range analysis technique which is particularly for the hulls in our problem. Second, our method still suffers from the curse of dimensionality in the worst case, although the boundary-aware subdivision helps a lot on avoiding subdividing the non-critical regions. This aspect can be improved by considering symbolic representations of the hulls, such as Taylor models Huang et al. [2022].

Besides, we observed that some hulls only have a very small intersection with the boundary but requires many subdivisions to find purely safe or unsafe subsets. For such a hull, we consider to develop shrinking methods that can fast find subsets by contracting the original hull.

6 CONCLUSION

We present a novel approach of probabilistic verification of DNNs and the main idea is to efficiently find safe or unsafe probabilistic hulls using a boundary-aware subdivision strategy. Our method does not have a limitation to particular activation functions and shows a much better performance than the similar existing techniques.

In the future, we seek to improve the method by (i) developing more efficient output range computation techniques particularly for the probabilistic hulls, to make the verification step less time-consuming; (ii) designing new symbolic representations for the hulls such that they are not limited to boxes or parallelotopes; and (iii) shrinking undecided hulls via interval constraint propagation, iteratively reducing the hull in each dimension until its image no longer intersects the safety boundary

Table 4: Efficiency of Finding Probabilistic Hulls (parallel). Legend - Time: runtime in seconds, End: end time = 1800 seconds, Others are the same as those in Table 1.

Net	Approach	L_s	U_s	$U_s - L_s$	Time (s)
1-6	Ours	**0.9815**	**0.9995**	**0.0180**	**264.00**
1-6	BaB	0.9397	1.0000	0.0603	End
2-2	Ours	**0.9819**	**0.9997**	**0.0178**	**142.30**
2-2	BaB	0.9616	1.0000	0.0384	End
2-9	Ours	**0.9502**	**0.9994**	**0.0492**	**686.66**
2-9	BaB	0.8896	1.0000	0.1104	End
3-1	Ours	**0.9865**	1.0000	**0.0135**	**68.06**
3-1	BaB	0.9845	1.0000	0.0155	1413.22
3-6	Ours	**0.9293**	**0.9976**	**0.0682**	**1188.63**
3-6	BaB	0.7738	1.0000	0.2262	End
3-7	Ours	**0.8673**	**0.9448**	**0.0775**	**1338.55**
3-7	BaB	0.7682	1.0000	0.2318	End
4-1	Ours	**0.9864**	1.0000	**0.0136**	**59.52**
4-1	BaB	0.9837	1.0000	0.0163	1019.67
4-7	Ours	**0.9778**	1.0000	**0.0222**	**397.20**
4-7	BaB	0.8993	1.0000	0.1007	End
5-3	Ours	**0.9823**	1.0000	**0.0177**	**303.09**
5-3	BaB	0.8886	1.0000	0.1114	End

References

Lee J. Bain and Max Engelhardt. *Introduction to Probability and Mathematical Statistics (2nd ed.)*. Cengage Learning, 1992.

David Boetius, Stefan Leue, and Tobias Sutter. Solving probabilistic verification problems of neural networks using branch and bound. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 4660–4699. PMLR, 2025.

Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, Nicola Paoletti, Andrea Patane, and Matthew Wicker. Statistical guarantees for the robustness of bayesian neural networks. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 5693–5700, 2019.

Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. Knowledge distillation: A survey. *Int. J. Comput. Vis.*, 129(6):1789–1819, 2021.

Chao Huang, Jiameng Fan, Xin Chen, Wenchao Li, and Qi Zhu. POLAR: A polynomial arithmetic framework for verifying neural-network controlled systems. In Ahmed Bouajjani, Lukás Holík, and Zhilin Wu, editors, *Proceedings of the 20th International Symposium on Automated Technology for Verification and Analysis*, volume 13505 of *Lecture Notes in Computer Science*, pages 414–430. Springer, 2022.

L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis*. Springer, 2001.

Guy Katz, Clark W. Barrett, David L. Dill, Kyle D. Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *The 29th International Conference on Computer Aided Verification (CAV 2017)*, volume 10426, pages 97–117. Springer, 2017.

Guido Manfredi and Yannick Jestin. An introduction to acas xu and the challenges ahead. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–9, 2016.

Ravi Mangal, Aditya V. Nori, and Alessandro Orso. Robustness of neural networks: a probabilistic and practical approach. In *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results, ICSE (NIER) 2019, Canada*, pages 93–96, 2019.

Mark Huasong Meng, Guangdong Bai, Sin Gee Teo, Zhe Hou, Yan Xiao, Yun Lin, and Jin Song Dong. Adversarial robustness of deep neural networks: A survey from a formal verification perspective. *CoRR*, abs/2206.12227, 2022.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill Education, 1997.

Joshua Pilipovsky, Vignesh Sivaramakrishnan, Meeko Oishi, and Panagiotis Tsiotras. Probabilistic verification of relu

neural networks via characteristic functions. In *Learning for Dynamics and Control Conference, L4DC 2023, 15-16 June 2023, Philadelphia, PA, USA*, volume 211 of *Proceedings of Machine Learning Research*, pages 966–979, 2023.

Florian Tambon, Gabriel Laberge, Le An, Amin Nikanjam, Paulina Stevia Nouwou Mindom, Yann Pequignot, Foutse Khomh, Giulio Antoniol, Ettore Merlo, and François Laviolette. How to certify machine learning based safety-critical systems? A systematic literature review. *Autom. Softw. Eng.*, 29(2):38, 2022.

Hoang-Dung Tran, Sungwoo Choi, Hideki Okamoto, Bardh Hoxha, Georgios Fainekos, and Danil V. Prokhorov. Quantitative verification for neural networks using probstars. In *Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2023, USA,*, pages 4:1–4:12, 2023.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Sekhar Jana, Cho-Jui Hsieh, and J. Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In *34th Annual Conference on Neural Information Processing Systems, (NeurIPS 2021)*, pages 29909–29921, 2021.

Stefan Webb, Tom Rainforth, Yee Whye Teh, and M. Pawan Kumar. A statistical approach to assessing neural network robustness. In *7th International Conference on Learning Representations, USA*, 2019.

Lily Weng, Pin-Yu Chen, Lam Nguyen, Mark Squillante, Akhilan Boopathy, Ivan Oseledets, and Luca Daniel. PROVEN: Verifying robustness of neural networks with a probabilistic approach. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6727–6736. PMLR, 2019.

Matthew Wicker, Luca Laurenti, Andrea Patane, Nicola Paoletti, Alessandro Abate, and Marta Kwiatkowska. Probabilistic reach-avoid for bayesian neural networks. *Artif. Intell.*, 334:104132, 2024a.

Matthew Wicker, Andrea Patane, Luca Laurenti, and Marta Kwiatkowska. Adversarial robustness certification for bayesian neural networks. In *Formal Methods - 26th International Symposium, FM 2024, Milan, Italy, September 9-13, 2024, Proceedings, Part I*, volume 14933 of *Lecture Notes in Computer Science*, pages 3–28, 2024b.

Haoze Wu, Omri Isac, Aleksandar Zeljic, Teruhiro Tagomori, Matthew L. Daggitt, Wen Kokke, Idan Refaeli, Guy Amir, Kyle Julian, Shahaf Bassan, Pei Huang, Ori Lahav, Min Wu, Min Zhang, Ekaterina Komendantskaya, Guy Katz, and Clark W. Barrett. Marabou 2.0: A versatile formal analyzer of neural networks. In *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part II*, volume 14682 of *Lecture Notes in Computer Science*, pages 249–264, 2024.

Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Xiaodong Yang, Tom Yamaguchi, Hoang-Dung Tran, Bardh Hoxha, Taylor T. Johnson, and Danil V. Prokhorov. Neural network repair with reachability analysis. In *Formal Modeling and Analysis of Timed Systems - 20th International Conference, FORMATS 2022, Warsaw, Poland, September 13-15, 2022, Proceedings*, volume 13465 of *Lecture Notes in Computer Science*, pages 221–236. Springer, 2022.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *31st Annual Conference on Neural Information Processing Systems 2018, (NeurIPS 2018)*, pages 4944–4953, 2018.

A HYPERPARAMETER AND GRID SEARCH

We performed an extensive grid search to determine the best configuration for the core components of our approach: the sampling strategy, the impurity scheduler, and the regression tree depth.

A.1 ACAS XU BENCHMARK

For the ACAS Xu benchmark, the search space was defined as follows.

- **Sampling Weights ($w_{\text{uniform}}, w_{\text{distribution}}$):** Five configurations were tested. $(0.0, 1.0)$, $(0.25, 0.75)$, $(0.5, 0.5)$, $(0.75, 0.25)$, and $(1.0, 0.0)$.
- **Regression Tree Depth:** Two values were considered. 5 and 10.
- **Impurity Splitting (β):** Five values were tested. $0, 0.25, 0.5, 0.75$, and 1.0.
- **Impurity Coefficient (α):** Six values were evaluated. $0.05, 0.1, 0.3, 0.5, 1.0$, and 2.0.

This parameter space resulted in **25** impurity configurations and a total of **250** parameter combinations. The evaluation metrics for each combination were the remaining unknown region probability ($U_s - L_s$) and the corresponding runtime.

The grid search was performed on **ACAS network 1-6 for specification 2**. Figure 4 presents the results, mapping the trade-off between $U_s - L_s$ (vertical axis) and runtime (horizontal axis). Each point represents a distinct configuration. The red star highlights the **Pareto optimal configuration**, which outperforms all other combinations in both metrics. Gray points indicate configurations that are dominated by the Pareto solution in at least one metric.

The search identified the optimal configuration for the ACAS benchmark as pure distributional sampling $(0.0, 1.0)$, a regression tree depth of 5, $\beta = 0.75$, and $\alpha = 0.05$. These settings were subsequently adopted for the main experiments on the ACAS Xu benchmark.

A.2 ROCKET LANDER BENCHMARK

The same systematic grid search methodology was applied to the Rocket Lander controller. The search space for this benchmark was defined as follows.

- **Sampling Weights ($w_{\text{uniform}}, w_{\text{distribution}}$).** The same five configurations were tested. $(0.0, 1.0)$, $(0.25, 0.75)$, $(0.5, 0.5)$, $(0.75, 0.25)$, and $(1.0, 0.0)$.
- **Regression Tree Depth.** Three values were considered. 5, 10, and 20.

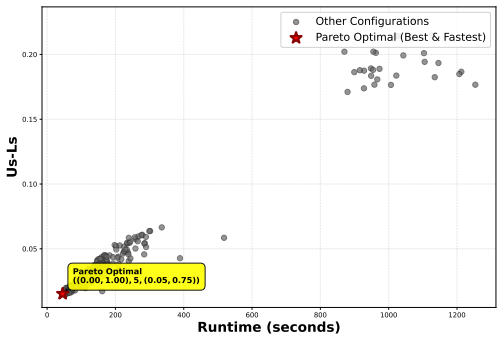

Figure 4: Scatter plot showing the relationship between Us-Ls and runtime for 250 different parameter configurations tested in the grid search.

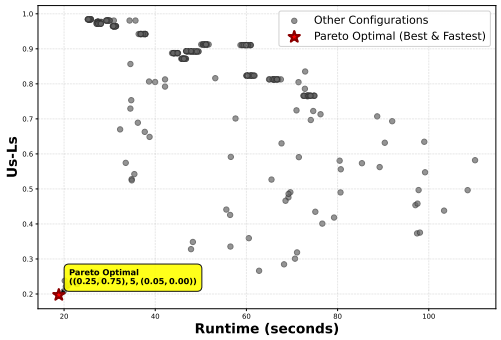

Figure 5: Scatter plot showing the relationship between Us-Ls and runtime for 300 different parameter configurations tested in the grid search.

- **Impurity Splitting (β).** Five values were tested. $0, 0.25, 0.5, 0.75$, and 1.0.
- **Impurity Coefficient (α).** Five values were evaluated. $0.05, 0.1, 0.3, 0.5$, and 1.0.

This yielded a total of **300** parameter combinations. The grid search was performed on **Rocket Net 1 for specification 1**.

Figure 5 presents the corresponding Pareto front analysis. Consistent with the ACAS results, the $U_s - L_s$ is plotted against the runtime, and the red star marks the Pareto optimal configuration.

The optimal configuration determined for the Rocket Lander benchmark was a *mixed sampling strategy $(0.25, 0.75)$, a regression tree depth of 5, $\beta = 0.0$, and $\alpha = 0.05$**. These settings were adopted for the main experiments on the Rocket Lander benchmark.

B ABLATION STUDIES

We conduct a comprehensive ablation study to analyze the independent contribution and combinatorial effects of the

key components in our proposed approach. The components investigated include the weighted mixed sampling and the impurity scheduler defined by α and β in our hyperparameter settings. The ablation studies are conducted on both the high-dimensional Rocket Lander and the moderate-dimensional ACAS Xu benchmarks. The following sections detail the findings, quantifying the impact of each mechanism on verification tightness ($U_s - L_s$) and time efficiency across these diverse benchmarks.

B.1 CONTRIBUTION OF SAMPLING AND IMPURITY SCHEDULING

The analysis of results presented in Table 5 focuses on the core question of whether sampling alone is effective and how the Impurity Scheduler contributes to the final performance. The data reveals that the **Impurity Scheduler** is the critical component for achieving high verification accuracy ($U_s - L_s$).

When the Impurity Scheduler is disabled, varying the sampling ratio only provides marginal improvements in $U_s - L_s$. For the Rocket Lander, the best sampling ratio ($w_u = 0.25$ or $w_u = 0.50$) achieves 0.9124, barely better than the worst sampling ratio (0.9194). For ACAS Xu, the sampling variance shows negligible impact on accuracy.

However, enabling the Impurity Scheduler changes the performance profile. Firstly, for the pure uniform sampling case ($w_u = 1.00$), $U_s - L_s$ is reduced from 0.9194 to 0.2466 for Rocket Lander and from 0.0650 to 0.0341 for ACAS Xu. This confirms that the scheduler can significantly tighten the verification bounds.

Furthermore, when the Impurity Scheduler is active, optimizing the mixed sampling ratio becomes relevant, providing a measurable gain. For both benchmarks, the $(\mathbf{0.75}, \mathbf{0.25})$ mixed ratio achieves the tightest overall bound (0.2371 for Rocket Lander and 0.0340 for ACAS Xu). This indicates that while sampling is not the main driver of accuracy, an optimal mix of uniform and distributional sampling is necessary for the final refinement.

Regarding runtime efficiency, the runtime for nearly all configurations, remains within a similar order of magnitude on both benchmarks. For instance, the Rocket Lander configurations generally fall between 21s and 31s, and the ACAS Xu configurations are mostly between 210s and 280s. This suggests that the major performance contribution of the Impurity Scheduler lies in the massive improvement of **accuracy** ($U_s - L_s$) convergence, while overall execution time remains relatively constant across effective configurations.

B.2 ABLATION ON IMPURITY SCHEDULER THRESHOLD

Given the critical role of the Impurity Scheduler, we conduct a focused ablation study by varying the threshold β, which controls the strategy for transitioning from standard longest-dimension splitting to the proposed soft-coded impurity measure (α-weighted splitting). The results, using pure uniform sampling ($w_u = 1.00$), are summarized in Table 6. The optimal transition strategy is found to be benchmark-dependent.

For Rocket Lander benchmark, the critical finding is that the **Immediate Soft Impurity** ($\beta = 0.00$) mode is optimal. This configuration achieves the tightest bound of $\mathbf{0.2466}$ and the best runtime of $\mathbf{23.17}$s within this set. This suggests that the longest-dimension splitting provides minimal structural benefit, and immediate application of the boundary-aware measure is key.

As for ACAS Xu, a **hybrid splitting strategy** is essential. The Immediate Soft Impurity ($\beta = 0.00$) yields a poor $U_s - L_s$ of 0.0610 and the longest runtime of 349.11s. In contrast, the best accuracy ($\mathbf{0.0341}$) is achieved when $\beta = 0.75$ (Late Activation), allowing the longest-dimension splitting to run for the initial portion of the verification. The best runtime ($\mathbf{195.23}$s) is achieved at $\beta = 0.50$ (Mid Activation). This confirms that for ACAS Xu, initial global refinement via standard splitting significantly accelerates the process before the fine-grained, boundary-aware splits are activated.

The distinct optimal β values can be explained by the fundamental differences in input dimensionality and the subsequent geometric challenges presented to the verification method (which relies on convex hull approximations):

1. **Geometric Control:** In methods utilizing convex hull approximations, the Longest-Dimension splitting is employed primarily to maintain a **low inter-dimension ratio** for the divided regions. Preventing regions from becoming highly elongated is vital, as this mitigates the over-approximation error that occurs when hull methods are applied to non-square regions.
2. **High Dimensionality (Rocket Lander):** The 9-dimensional space of Rocket Lander limits the effectiveness of pure Longest-Dimension splitting. The soft impurity measure from the beginning is necessary because its geometric penalty term (L^α) simultaneously guides the split toward reducing boundary uncertainty (via MSE) and acts to regularize the region's aspect ratio by penalizing splits across large dimensions (L). This combined efficiency makes immediate activation optimal, as the structural splitting stage provides little benefit.
3. **Initial Refinement Need (ACAS Xu):** In the lower 5-dimensional ACAS Xu space, the Longest-Dimension

Table 5: Ablation Study on Rocket Lander and ACAS Xu (Average Results). The table compares the average unknown probability ($U_s - L_s$) and average runtime for different component configurations on two benchmarks. The weights are w_u (uniform) and w_d (distributional).

Configuration	(w_u, w_d)	**Rocket Lander**		**ACAS Xu**	
		Avg. $U_s - L_s$	**Avg. Time (s)**	**Avg.** $U_s - L_s$	**Avg. Time (s)**
Baseline Configurations (Impurity Scheduler Disabled)					
Baseline	(1.00, 0.00)	0.9194	30.26	0.0650	279.46
	(0.00, 1.00)	0.9242	24.86	0.0681	279.05
	(0.75, 0.25)	0.9134	30.74	0.0652	277.95
	(0.50, 0.50)	0.9124	29.22	0.0664	279.09
	(0.25, 0.75)	0.9124	27.80	0.0662	278.14
Configurations with Impurity Scheduler Enabled					
+ Impurity Scheduler	(1.00, 0.00)*	0.2466	23.17	0.0341	211.61
	(0.00, 1.00)	0.3294	25.23	0.0368	221.05
	(0.75, 0.25)	**0.2371**	**21.25**	**0.0340**	**210.33**
	(0.50, 0.50)	0.2705	22.02	0.0344	214.28
	(0.25, 0.75)	0.2576	21.71	0.0350	218.96

*For ACAS Xu, the impurity scheduler uses $\beta = 0.75$ and $\alpha = 0.05$ as the representative Impurity-only configuration for comparison. For Rocket Lander, the impurity scheduler uses $\beta = 0.0$ and $\alpha = 0.05$ as the representative Impurity-only configuration for comparison.

Table 6: Ablation Study on Impurity Scheduler Threshold (β). The table compares the performance ($U_s - L_s$ and Runtime) of the Impurity Scheduler by varying the threshold β, which controls the transition from longest-dimension splitting ($\beta = 1.0$) to soft-coded impurity splitting ($\beta < 1.0$). Only pure uniform sampling ($w_u = 1.00$) is used.

Threshold β	**Splitting Mode**	**Rocket Lander**		**ACAS Xu**	
		Avg. $U_s - L_s$	**Avg. Time (s)**	**Avg.** $U_s - L_s$	**Avg. Time (s)**
Baseline (β Implied)	Longest-Dimension	0.9194	30.26	0.0650	279.46
0.75	Late Activation	0.9194	30.24	**0.0341**	211.61
0.50	Mid Activation	0.9194	30.53	0.0342	**195.23**
0.25	Early Activation	0.9194	30.29	0.0378	210.71
0.00	Immediate Soft Impurity	**0.2466**	**23.17**	0.0610	349.11

splitting remains an efficient method for achieving an initial, coarse, global subdivision. This initial phase (enabled by $\beta = 0.75$ or $\beta = 0.50$) establishes regions with more uniform aspect ratios. This preprocessing ensures that when the fine-grained soft impurity measure is activated later, it operates on geometrically well-behaved regions, thereby improving both accuracy and overall convergence speed.

This analysis confirms that our impurity scheduler is a highly effective component, but its optimal configuration requires careful tuning based on the specific dimensional and structural characteristics of the benchmark.

C RUNTIME BREAKDOWN

The runtime of our method concentrates in a few components, and we break each down to show the bottleneck. As in the main paper, L_s is the lower bound on safe probability and $U_s - L_s$ the unknown interval. We measure wall-clock time for the five components reported in Table 7: **sampling** (boundary-aware sampling, both initial and iterative), **tree construction** (regression-tree splitting search), **CROWN** (bound computation via `auto_LiRPA`), **data generation** (forward passes that produce training labels), and **Other**. The *Other* bucket is the residual after subtracting the four explicitly measured components from the total wall-clock time. It covers overheads not profiled by the four main timing blocks: the final aggregation step (probability totals, output assembly), the portion of region extraction not attributed to CROWN (constructing input boxes from tree leaves, computing per-region probability mass, accumulating global statistics), and any unprofiled code between timing checkpoints. In our runs, the region-extraction overhead dominates *Other*, and final aggregation alone is negligible (under 0.02 s). Three ACAS Xu networks span easy,

medium, and hard cases under Property P2, with the same hyperparameters as Table 1: pure distributional sampling, depth 5, $\beta = 0.75$, $\alpha = 0.05$. We report both modes: parallel (CROWN batched into one call) and serial (CROWN called at every tree split).

CROWN dominates every case: 50–59% in parallel mode and roughly 90% in serial mode, with parallel batching giving a stable 3.8× speedup. Sampling stays below 4% and data generation below 1%. Tree construction is 18–21% in parallel mode. A cheaper output-range analysis tailored to probabilistic hulls would directly reduce the dominant CROWN cost.

D CONTROLLED SCALING EXPERIMENTS

The main benchmark reports a single number per (network, property) pair, so width and input dimensionality cannot be separated. Three controlled experiments vary one axis of difficulty at a time, allowing cost changes to be attributed to a single factor. All use the same hyperparameters as the main runs (parallel mode, 1000 initial samples, 100 iterative, depth 5, $\varepsilon = 10^{-5}$). In all tables, L_s is the lower bound on safe probability and $U_s - L_s$ is the unknown probability interval.

D.1 HIDDEN-WIDTH SCALING

We test two hypotheses on what drives hull-generation cost: a *parameter-driven* hypothesis (runtime scales with ReLU node count) and a *geometry-driven* hypothesis (runtime scales with decision-boundary complexity, not parameter count). A monotonic time-vs-width curve would confirm the first; a non-monotonic curve would confirm the second.

We distill ReLU students at widths $w \in \{10, 20, 25, 60, 100\}$ from three ACAS Xu teachers: `1-6` (easy), `4-1` (medium), and `3-7` (hard). All students keep the original 6-hidden-layer architecture (original width: 50 neurons per layer); only the per-layer width changes. Distillation uses a KL+MSE loss (T=30, α=0.05) for 500–2000 epochs on 20,000 uniform samples. Classification agreement with the teacher ranges from 0.57 (w=10 on 1-6) to 0.95 (w=100 on 1-6). Table 9 reports the verification cost for all 15 (teacher, width) configurations.

Width is not monotonic, refuting the parameter-driven hypothesis. On teacher 3-7, w=25 is the hardest distilled configuration (197.3 s, 2889 iterations), while w=60 (161.0 s) and w=100 (68.1 s) are easier.

Teacher identity predicts difficulty better than width: all distilled networks from teacher 3-7 require more verification effort than any distilled network from 1-6, and most exceed those from 4-1, though the hardest 4-1 student (w=60, 85.4 s) exceeds the easiest 3-7 student (w=10, 41.5 s). Distillation also reduces verification cost compared to the original networks. For teacher 1-6, the largest distilled student (w=100, 13.7 s) is faster than the original (46.6 s); for teacher 3-7, even the hardest distilled student (w=25, 197.3 s) is faster than the original (379.9 s).

In summary, parameter count alone does not determine verification cost; what matters is the geometry of the learned decision boundary.

D.2 INACTIVE-INPUT-DIM WIDTH SCALING

The main benchmark uses a fixed 5-dim input, so the effect of input dimensionality is not tested. We ask whether cost is driven by the *number* of extra dimensions or by their *width*, since in deployment both sensor noise and adversarial perturbations can add dimensions whose width matches or exceeds the spec range.

We add one or more *inactive* dimensions to the network input. The original 5 active dimensions ($\rho, \theta, \psi, v_{\text{own}}, v_{\text{int}}$) are kept as they are. A zero-pad wrapper slices off the inactive dimensions before forwarding, so the network function is unchanged, but the search now covers a higher-dimensional space. Each inactive dimension is assigned a Gaussian whose (lb, ub, μ, σ) are copied from one of the five active dimensions. In this subsection we fix d=6 (one inactive dimension) and choose its template from three options that span the active-dim widths: *min* (v_{own}, 0.050), *median* (ρ, 0.080), and *max* (ψ, 0.9995). We refer to these conditions as $d_{6,\text{min}}$, $d_{6,\text{med}}$, and $d_{6,\text{max}}$. The d_5 baseline is the unmodified 5-dim network.

We first run the four conditions on `1-6` to establish the effect, then repeat on four more networks (`5-3`, `4-1`, `3-1`, `2-2`) to check generalization. The four added networks cover the full difficulty range and all five cell rows of the ACAS grid. The d_5 baseline L_s and $U_s - L_s$ values are from the paper's published *Ours* column; the d_5 runtimes are fresh measurements from the same codebase and hardware as the d_6 runs. Table 10 reports the data.

On `1-6`, one inactive dimension of width ≈ 1.0 increases runtime from 55.7 s to 612.4 s (11.0×) and $U_s - L_s$ from 0.0154 to 0.0707 (4.6×), while one inactive dimension of width 0.05 increases runtime to 73.6 s (1.3×) and $U_s - L_s$ to 0.0194 (1.3×). The cost is monotonic in the width of the inactive dimension.

The effect is consistent across all five networks: the $d_5 > d_{6,\text{min}} > d_{6,\text{med}} > d_{6,\text{max}}$ ordering in L_s holds with no reversals. The hardest networks degrade the most: on `2-2` (the hardest), $d_{6,\text{max}}$ times out at 1800 s; on `4-1` and `3-1`, $d_{6,\text{max}}$ converges but at high cost (1386.8 s and 1269.1 s), while $d_{6,\text{med}}$ remains in budget.

Table 7: Runtime breakdown in parallel mode on three ACAS Xu networks (Property P2). Columns show absolute time (s) and percentage of total wall-clock time. $U_s - L_s$ is the unknown-probability interval; Iterations is the number of tree-split refinement steps.

Net	**5-3**		**4-1**		**3-7**	
	(s)	(%)	(s)	(%)	(s)	(%)
Sampling	1.92	2.3%	5.05	3.2%	11.63	3.1%
Tree construction	17.19	20.7%	28.35	17.8%	68.34	18.0%
CROWN (bound comp.)	47.73	57.6%	93.25	58.7%	190.00	50.0%
Data generation (fwd)	0.49	0.6%	0.96	0.6%	1.93	0.5%
Other	15.54	18.8%	31.38	19.7%	107.97	28.4%
Total	**82.87**	**100%**	**158.99**	**100%**	**379.87**	**100%**
$U_s - L_s$	0.0266		0.0383		0.0763	
Iterations	1,713		3,326		6,695	

Table 8: Runtime in serial mode and parallel speedup on the same three networks. *CROWN Calls* is the number of `auto_LiRPA` invocations. *Speedup* is the ratio serial time / parallel time. The CROWN share in serial mode is a lower bound because CROWN is invoked inside tree construction in this mode.

Net	**Total (s)**	**CROWN (s)**	**CROWN %**	**CROWN Calls**	**Speedup**
5-3	310.71	284.04	91.4%	10,458	3.7×
4-1	601.04	544.64	90.6%	19,799	3.8×
3-7	1474.58	1317.92	89.4%	48,333	3.9×

D.3 INPUT-DIM-COUNT SCALING

The previous subsection isolated *width* at a fixed count of one extra dimension. The complementary question is: at a fixed *width*, how does the cost scale with the *count* of inactive dimensions? The width-by-count interaction determines the verifier's robustness to realistic sensor noise, where both noise width and the number of noisy sensors are deployment parameters.

We extend the controlled-template design to higher input dimensions: $d \in \{6, 7, 8, 9, 10\}$ (one to five extra inactive dimensions), again with three templates (*min*, *median*, *max*) and the `1-6` network. The d=6 row reuses the (1-6) values from Table 10. All extra dimensions use the same template per condition, so the inactive-dim width is held fixed while the count grows. Table 11 reports runtime and L_s against d for each of the three templates.

The data confirm the width-by-count interaction.

When inactive dimensions are narrow, adding more of them is cheap. Going from d=5 to d=10 with the *min* template (extra widths 0.05) costs $3\times$ more time (54.7 s to 166.5 s) and reduces L_s by only 0.016 ($0.985 \to 0.969$).

Width dominates count. With the *max* template ($w \approx 1.0$), all four $d \in \{7, 8, 9, 10\}$ configurations time out at 1800 s. L_s drops steadily (0.69, 0.47, 0.25, 0.13); by d=10, the verifier returns a near-trivial bound ($U_s - L_s = 0.87$, close to the worst-case interval of 1). One wide inactive dimension at d=6 (612.4 s) is harder than five narrow ones at d=10 (166.5 s).

The *median* template interpolates between the two extremes: runtime grows with d (116 s at d=6 to 844 s at d=10), and L_s drops monotonically ($0.979 \to 0.910$).

D.4 SUMMARY

The three scaling experiments show a consistent pattern: the verifier's cost is driven by the geometry of the search space—axis-aligned splits in wide directions—not by the network's parameter count. Distillation consistently reduces verification cost across the three tested teachers, consistent with the hypothesis that distilled networks learn smoother decision boundaries. Adding inactive dimensions of small width is nearly free, but adding even one inactive dimension of width comparable to the existing active dimensions can multiply runtime by 4–$11\times$; with the widest template at d=10, the verifier returns a near-trivial bound ($L_s = 0.13$, $U_s - L_s = 0.87$). The practical recommendation is to keep the input distribution tight in every dimension, which motivates geometry-aware splitting heuristics as future work.

Table 9: Hidden-width scaling: verification runtime and tightness across three teachers and five distilled widths.

Teacher	Width	Time (s)	Iterations	$U_s - L_s$
Teacher: ACAS 1-6 (easy)				
ReLU distill	10	1.5	12	0.013
ReLU distill	20	3.2	49	0.014
ReLU distill	25	2.7	44	0.013
ReLU distill	60	3.5	59	0.013
ReLU distill	100	13.7	221	0.014
Teacher: ACAS 4-1 (medium)				
ReLU distill	10	0.1	2	0.013
ReLU distill	20	1.5	27	0.013
ReLU distill	25	6.7	105	0.014
ReLU distill	60	85.4	1246	0.032
ReLU distill	100	14.1	241	0.014
Teacher: ACAS 3-7 (hard)				
ReLU distill	10	41.5	751	0.020
ReLU distill	20	49.1	853	0.018
ReLU distill	25	197.3	2889	0.049
ReLU distill	60	161.0	2328	0.041
ReLU distill	100	68.1	1015	0.029

Table 10: Inactive-input-dim width scaling across 5 ACAS networks (P2). Each network has four conditions: the d_5 baseline (from the paper) and $d_{6,\text{min}}$ / $d_{6,\text{med}}$ / $d_{6,\text{max}}$ with the inactive dimension's (lb, ub, μ, σ) copied from the named active dimension. T.O. marks a run that timed out at 1800 s. Iteration counts are reported for the own runs; the d_5 paper baselines do not report per-condition iterations.

Network	Condition	Inactive template	L_s	$U_s - L_s$	Time (s)	Iterations
1-6 (easy)	d_5	—	0.9846	0.0154	55.7	—
	$d_{6,\text{min}}$	v_own (w=0.0500)	0.9806	0.0194	73.6	1301
	$d_{6,\text{med}}$	rho (w=0.0799)	0.9788	0.0212	115.7	1989
	$d_{6,\text{max}}$	psi ($w\approx$1.0)	0.9293	0.0707	612.4	9414
5-3	d_5	—	0.9754	0.0246	85.6	—
	$d_{6,\text{min}}$	v_own (w=0.0500)	0.9696	0.0304	97.0	1716
	$d_{6,\text{med}}$	rho (w=0.0799)	0.9659	0.0341	154.3	2117
	$d_{6,\text{max}}$	psi ($w\approx$1.0)	0.9329	0.0671	396.6	6394
4-1	d_5	—	0.9645	0.0355	218.2	—
	$d_{6,\text{min}}$	v_own (w=0.0500)	0.9528	0.0471	265.7	4265
	$d_{6,\text{med}}$	rho (w=0.0799)	0.9427	0.0573	402.8	5718
	$d_{6,\text{max}}$	psi ($w\approx$1.0)	0.8325	0.1675	1386.8	16360
3-1	d_5	—	0.9240	0.0476	289.3	—
	$d_{6,\text{min}}$	v_own (w=0.0500)	0.9198	0.0542	317.8	3500
	$d_{6,\text{med}}$	rho (w=0.0799)	0.9135	0.0638	349.7	4234
	$d_{6,\text{max}}$	psi ($w\approx$1.0)	0.8628	0.1338	1269.1	11240
2-2 (hard)	d_5	—	0.9250	0.0750	406.9	—
	$d_{6,\text{min}}$	v_own (w=0.0500)	0.9171	0.0775	469.0	4795
	$d_{6,\text{med}}$	rho (w=0.0799)	0.9092	0.0879	551.1	5839
	$d_{6,\text{max}}$	psi ($w\approx$1.0)	0.8257	0.1743	T.O.	13815

Table 11: Dim-count scaling with controlled inactive-dim width (ACAS `1-6`, P2). All $d - 5$ extra dimensions use the same template, whose (lb, ub, μ, σ) are copied from the named active dimension. T.O. marks a run that timed out at 1800 s; iteration counts for T.O. runs are partial (terminated before convergence).

Template	**Width**	d	L_s	$U_s - L_s$	**Time (s)**	**Iterations**
— (baseline)	—	5	0.9847	0.0153	54.7	985
min	0.0500	6	0.9806	0.0194	73.6	1301
min	0.0500	7	0.9783	0.0217	108.0	1443
min	0.0500	8	0.9733	0.0267	132.5	1590
min	0.0500	9	0.9707	0.0293	140.9	1834
min	0.0500	10	0.9689	0.0311	166.5	1725
median	0.0799	6	0.9788	0.0212	115.7	1989
median	0.0799	7	0.9722	0.0278	265.1	3343
median	0.0799	8	0.9643	0.0357	324.0	3873
median	0.0799	9	0.9382	0.0618	692.8	7653
median	0.0799	10	0.9103	0.0897	843.9	9006
max	0.9995	6	0.9293	0.0707	612.4	9414
max	0.9995	7	0.6910	0.3090	T.O.	17331
max	0.9995	8	0.4708	0.5292	T.O.	18031
max	0.9995	9	0.2518	0.7482	T.O.	18192
max	0.9995	10	0.1317	0.8683	T.O.	19704