RABBit: Precise Relational DNN Verification With Cross Executional Branching

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

We propose RABBit, a Branch-and-Bound-based verifier for verifying relational properties defined over Deep Neural Networks, such as robustness against universal adversarial perturbations (UAP). Existing SOTA complete L_∞-robustness verifiers can not reason about dependencies between multiple executions and, as a result, are imprecise for relational verification. In contrast, existing SOTA relational verifiers only apply a single bounding step and do not utilize any branching strategies to refine the obtained bounds, thus producing imprecise results. We develop the first scalable Branch-and-Bound-based relational verifier, RABBit, which efficiently combines branching over multiple executions with cross-executional bound refinement to utilize relational constraints, gaining substantial precision over SOTA baselines on a wide range of datasets and networks.

1 Introduction

Deep neural networks (DNNs) are now widely used in safety-critical fields like autonomous driving and medical diagnosisAmato et al. [2013], where their decisions can have serious consequences. However, understanding and ensuring their reliability is difficult due to their complex and opaque nature. Despite efforts to find and address vulnerabilities, such as adversarial attacks Goodfellow et al. [2014], Madry et al. [2018], Moosavi-Dezfooli et al. [2017], Potdevin et al. [2019], Wu et al. [2023b], Sotoudeh and Thakur [2020] and adversarial training techniques Madry et al. [2018], ensuring safety remains a challenge. As a result, extensive research is focused on formally verifying the safety of DNNs. However, most of the existing L_∞ robustness verification techniques can not handle relational properties common in practical situations. While significant efforts have been invested in verifying the absence of input-specific adversarial examples within the local neighborhood of test inputs, recent studies Li et al. [2019a] emphasize that input-specific attacks are impractical regardless. Conversely, practical attack scenarios Liu et al. [2023], Li et al. [2019b,a] involve the creation of universal adversarial perturbations (UAPs) Moosavi-Dezfooli et al. [2017], which are crafted to impact a substantial portion of inputs from the training distribution. RACoon Banerjee and Singh [2024] showed that since the same adversarial perturbation is applied to multiple inputs, the executions on different perturbed inputs are related, exploiting the relationship between different executions significantly improves the precision of the verifier. Despite RACoon's ability to leverage cross-executional dependencies, RACoon remains imprecise as it only applies a single bounding step and lacks refinement using branching strategies used in SOTA complete non-relational verifiers.

Key challenges: For precise relational verification, we need efficient algorithms that can effectively combine branching strategies over multiple executions with bounding techniques that can leverage cross-executional dependencies. Theoretically, MILP (Mixed Integer Linear Programming) can exactly encode DNN executions with piecewise linear activation functions like ReLU over any input regions specified by linear inequalities. However, the associated MILP optimization problem is computationally expensive. For instance, encoding k executions of a DNN with n_r ReLU activations introduces $O(n_r \times k)$ integer variables in the worst case. As the cost of MILP optimization grows exponentially with the number of integer variables, even SOTA off-the-shelf solvers like Gurobi Gurobi Optimization, LLC [2018] struggle to verify small DNNs for a relational property over k executions within a reasonable time limit. For scalability, SOTA non-relational verifiers like α, β-CROWN Wang et al. [2021b] design custom "Branch and Bound" (BaB) solvers using more scalable differentiable optimization techniques such as gradient descent. However, these verifiers ignore dependencies between multiple executions, resulting in imprecise relational verification. Conversely, the SOTA relational verifier RACoon uses parametric linear relaxation for each activation to avoid integer variables and employs gradient descent to learn parameters that leverage cross-executional dependencies for verification. This method, however, introduces imprecision due to the replacement of non-linear activations with parametric linear approximations. Therefore, precise relational verification requires scalable algorithms that can: a) utilize cross-executional dependencies, b) effectively reduce imprecision from parametric linear relaxations, and c) scale to the large DNNs used in this paper.

Our contributions: We advance the state-of-the-art in relational DNN verification by:

- Efficiently combining branching strategies over multiple DNN executions with cross-executional bounding method that utilizes dependencies between DNN's outputs from different executions while reducing imprecision resulting from parametric linear relaxations.
- Developing two "branch and bound" algorithms, each with its own advantages - **a) strong bounding:** applies cross-execution bounding at each step, branching over all executions. This method provides tighter bounds than RACoon (cross-executional bound refinement without branching) and α, β-CROWN (branching without cross-executional bound refinement), **b) strong branching:** applies cross-execution bounding only at the start to derive fixed linear approximations for each execution. These approximations are then used to branch independently over each execution, exploring more branches per execution.
- Combining strong bounding and branching results into an efficiently optimizable MILP instance that leverages the benefits of both techniques, outperforming each individually.
- Performing extensive experiments on popular datasets and various DNNs (standard and robustly trained) to showcase the precision improvement over the current SOTA baselines.

2 Related Works

Non-relational DNN verifiers: DNN verifiers are broadly categorized into three main categories - (i) sound but incomplete verifiers which may not always prove property even if it holds Gehr et al. [2018], Singh et al. [2018, 2019b,a], Zhang et al. [2018], Xu et al. [2020, 2021], (ii) complete verifiers that can always prove the property if it holds Wang et al. [2018], Gehr et al. [2018], Bunel et al. [2020a,c], Bak et al. [2020], Ehlers [2017], Ferrari et al. [2022], Fromherz et al. [2021], Wang et al. [2021a], Palma et al. [2021], Anderson et al. [2020], Zhang et al. [2022a] and (iii) verifiers with probabilistic guarantees Cohen et al. [2019], Li et al. [2022].

Relational DNN verifier: DNN relational verifiers fall into two main categories: (i) verifiers for properties such as UAP and fairness, defined over multiple executions of the same DNN Zeng et al. [2023], Khedr and Shoukry [2023], Banerjee and Singh [2024], and (ii) verifiers for properties like local DNN equivalence, defined over multiple executions of different DNNs on the same input Paulsen et al. [2020, 2021]. For relational properties defined over multiple executions of the same DNN the existing verifiers Khedr and Shoukry [2023] reduce the verification problem into L_∞ robustness problem by constructing "product DNN" with multiple copies of the same DNN. However, the relational verifier in Khedr and Shoukry [2023] treats all k executions of the DNN as independent and loses precision as a result of this. Zeng et al. [2023] (referred to as I/O formulation) although tracks the relationship between inputs used in multiple executions at the input layer, does not track the relationship between the inputs fed to the subsequent hidden layers and can only achieve a limited improvement over the baseline verifiers that treat all executions independently. The SOTA relational verifier RACoon Banerjee and Singh [2024] improves relational verification's precision by leveraging cross-executional dependencies at all layers and introducing a new bounding strategy called cross-executional bound refinement, as detailed in Section 3. There exist, probabilistic verifiers, Xie et al. [2021], Zhang et al. [2022b] based on randomized smoothing Cohen et al. [2019] for verifying relational properties. However, these works can only give probabilistic guarantees on smoothed models which have high inference costs. Similar to Zeng et al. [2023], Banerjee and Singh [2024], in this work, we focus on deterministic relational verifiers for DNNs with ReLU activation.

However, RABBit can be extended to activations like Sigmoid, Tanh, etc. with branching methods from Shi et al. [2024] and parametric bounds from Wu et al. [2023a].

3 Preliminaries

We provide the necessary background on "branch and bound" (BaB) based non-relational DNN verification, as well as DNN safety properties that can be encoded as relational properties.

Non-relational DNN verification: For a single execution, non-relational DNN verification involves proving that the network outputs $\mathbf{y} = N(\mathbf{x} + \boldsymbol{\delta})$ for all perturbations $\mathbf{x} + \boldsymbol{\delta}$ of an input $\mathbf{x}$ specified by ϕ, satisfy a logical specification ψ. Common safety properties like L_∞ robustness encodes the output specification (ψ) as linear inequality (or conjunction of linear inequalities) over DNN output $\mathbf{y} \in \mathbb{R}^{n_l}$. e.g. $\psi(\mathbf{y}) = (\mathbf{c}^T\mathbf{y} \geq 0)$ where $\mathbf{c} \in \mathbb{R}^{n_l}$. In general, even for piece-wise linear activation functions and ϕ specified with linear inequalities complete DNN verification that always either proves the property or finds a counter-example is NP-hard. Instead, given a DNN $N : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l}$ and a property specified by (ϕ, ψ), scalable sound but incomplete verifiers compute a linear approximation specified by $\mathbf{L} \in \mathbb{R}^{n_0}, b \in \mathbb{R}$ such that for any input $\mathbf{x}$ satisfying ϕ the following condition holds $\mathbf{L}^T\mathbf{x} + b \leq \mathbf{c}^T N(\mathbf{x})$. For all $\mathbf{x}$ satisfying ϕ, the verifier then proves $\mathbf{L}^T\mathbf{x} + b \geq 0$, consequently showing $\mathbf{c}^T N(\mathbf{x}) \geq 0$. Although $\mathbf{L}^T\mathbf{x} + b$ always computes a valid lower bound on $\mathbf{c}^T N(\mathbf{x})$, it can be imprecise. Therefore, for piece-wise linear activations, SOTA non-relational verifiers apply a BaB method to improve precision. Each branching step decomposes the problem into multiple subproblems, while the bounding method computes a valid lower bound for each subproblem.
Branching for piecewise linear activation: The non-relational verifier computes $\mathbf{L}$ by replacing non-linear activations with linear relaxations, which introduces imprecision. However, for piecewise linear activations like ReLU, it is possible to consider each linear piece separately as different subproblems, avoiding the need for imprecise linear relaxations. For instance, for $y = ReLU(x)$, branching on x and considering the cases $x \leq 0$ and $x \geq 0$ allows decomposing $ReLU(x)$ into two distinct linear pieces. Still in the worst case decomposing all ReLU nodes in a DNN results in exponential blowup making it practically infeasible. Therefore, SOTA non-relation verifiers like α, β-CROWN Wang et al. [2021b] greedily pick a small subset of ReLU nodes for branching while using linear relaxations for the rest. We explain the bounding step used for each subproblem below.
Bounding with parameter refinement: Obtaining sound linear relaxations of activations σ like ReLU, which are not used for branching, involves computing linear lower bounds $\sigma_l(x)$ and upper bound $\sigma_u(x)$ that contain all possible outputs of σ w.r.t all inputs $\mathbf{x}$ satisfying ϕ. That is, for all possible input values x of σ, $\sigma_l(x) \leq \sigma(x) \leq \sigma_u(x)$ holds. SOTA non-relational verifiers, such as α, β-CROWN, improve precision by using parametric linear relaxations instead of static linear bounds and refine the parameters to facilitate verification of the property (ϕ, ψ). For example, for $ReLU(x)$, the parametric lower bound is $ReLU(x) \geq \alpha \times x$ with $\alpha \in [0, 1]$. Since $\alpha \times x$ remains a valid lower for any $\alpha \in [0, 1]$, this allows optimizing α while ensuring the bound remains mathematically correct. Each branched ReLU say $y = ReLU(x)$, introduces two subproblems each with one additional constraint $x \leq 0$ (or, $x \geq 0$) where ReLU behaves as a linear function i.e. $y = 0$ (or, $y = x$) respectively. To obtain the lower bound of $\mathbf{L}^T\mathbf{x} + b$ over inputs satisfying ϕ with the additional branching constraints α, β-CROWN convert the constrained optimization problem into an unconstrained one by looking at the Lagrangian dual. The dual replaces each branching constraint by augmenting the minimization objective $\mathbf{L}^T\mathbf{x} + b$ with additional terms i.e. $\mathbf{L}^T\mathbf{x} + b + \beta^+ x$ for $x \leq 0$ or $\mathbf{L}^T\mathbf{x} + b + \beta^- x$ for $x \geq 0$ where $\beta^+ \geq 0$ and $\beta^- \leq 0$. Overall, at high level, α, β-CROWN computes parametric linear approximations $\mathbf{L}(\boldsymbol{\alpha}, \boldsymbol{\beta})^T\mathbf{x} + b(\boldsymbol{\alpha}, \boldsymbol{\beta})$ and refine the parameters α, β to facilitate verification of (ϕ, ψ).

DNN relational properties: For a DNN $N : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l}$, relational properties defined over k executions of N are specified by the tuple (Φ, Ψ) where the input specification $\Phi : \mathbb{R}^{n_0 \times k} \to \{true, false\}$ encodes the input region $\Phi_t \subseteq \mathbb{R}^{n_0 \times k}$ encompassing all potential inputs corresponding to each of the k executions of N and the output specification $\Psi : \mathbb{R}^{n_l \times k} \to \{true, false\}$ specifies the safety property we expect the outputs of all k executions of N to satisfy. Formally, in DNN relational verification, given N, an input specification Φ and an output specification Ψ we require to prove whether $\forall \mathbf{x_1^*}, \ldots, \mathbf{x_k^*} \in \mathbb{R}^{n_0}.\Phi(\mathbf{x_1^*}, \ldots, \mathbf{x_k^*}) \implies \Psi(N(\mathbf{x_1^*}), \ldots N(\mathbf{x_k^*}))$ or provide a counterexample otherwise. Here, $\mathbf{x_1^*}, \ldots, \mathbf{x_k^*}$ are the inputs to the k executions of N and $N(\mathbf{x_1^*}), \ldots, N(\mathbf{x_k^*})$ are the corresponding outputs. Commonly, the input region ϕ_t^i for the i-th execution is a L_∞ region around a fixed point $\mathbf{x_i} \in \mathbb{R}^{n_0}$ defined as $\phi_t^i = \{\mathbf{x_i^*} \in \mathbb{R}^{n_0} \mid \|\mathbf{x_i^*} - \mathbf{x_i}\|_\infty \leq \epsilon\}$ while the corresponding output specification $\psi^i(N(\mathbf{x_i^*})) = \bigwedge_{j=1}^{m}(\mathbf{c_{i,j}}^T N(\mathbf{x_i^*}) \geq 0)$. Subsequently,

$\Phi(\mathbf{x_1^*}, \ldots, \mathbf{x_k^*}) = \bigwedge_{i=1}^{k} (\mathbf{x_i^*} \in \phi_t^i) \bigwedge \Phi^\delta(\mathbf{x_1^*}, \ldots, \mathbf{x_k^*})$ where $\Phi^\delta(\mathbf{x_1^*}, \ldots, \mathbf{x_k^*})$ encodes the relationship between the inputs used in different execution and $\Psi(N(\mathbf{x_1^*}), \ldots, N(\mathbf{x_k^*})) = \bigwedge_{i=1}^{k} \psi^i(N(\mathbf{x_i^*}))$. Next, we describe relational properties that can encode interesting DNN safety configurations.

UAP verification: In a UAP attack, given a DNN N, the adversary aims to find an adversarial perturbation with a bounded L_∞ norm that maximizes the rate at which N misclassifies when the same adversarial perturbation is applied to all inputs from the distribution. The UAP verification problem aims to find the worst-case accuracy of N against the UAP adversary. We refer to this worst-case accuracy as UAP accuracy in the rest of the paper. As shown by Theorem 2 in Zeng et al. [2023], it is possible to stastically estimate the UAP accuracy of N with respect to the input distribution if one can determine the UAP accuracy of N on k randomly selected images. We focus on the k-UAP verification problem for the rest of the paper as improving the precision of k-UAP verification directly improves the UAP accuracy on the input distribution Banerjee and Singh [2024]. The k-UAP verification problem is fundementally different from local L_∞ robustness verification since the same adversarial perturbation is applied across the set of inputs. Thus, improving precision for the UAP verification problem requires a relational verifier that can exploit dependencies between the perturbed inputs. We provide the Φ and Ψ of the UAP verification problem in Appendix A.1.

4 Cross-executional BaB

The key distinction between relational and non-relational DNN verification is the dependency between different DNN executions, which necessitates that any precise relational verifier utilizes these cross-execution dependencies. For instance, for k-UAP problem with two images $\mathbf{x_1}$, $\mathbf{x_2}$ consider the scenario where both $\mathbf{x_1}$ and $\mathbf{x_2}$ have valid adversarial perturbations $\delta_\mathbf{1}$ and $\delta_\mathbf{2}$ but no common perturbation say δ that works for both $\mathbf{x_1}$ and $\mathbf{x_2}$. In this case, any non-relational verification that does not account for cross-execution dependencies can never prove the absence of a common perturbation given that both $\mathbf{x_1}, \mathbf{x_2}$ have valid adversarial perturbations highlight the importance of utilizing cross-executional dependencies. The SOTA relational verifier RACoon Banerjee and Singh [2024] leverages cross-execution dependencies to **jointly** optimize the $\boldsymbol{\alpha}$ parameters from different executions, significantly improving the precision of relational verification. However, RACoon only uses parametric linear relaxations for non-linear activations and lacks a branching step, resulting in reduced precision, as confirmed by our experimental results in Section 6. To address this, we propose two separate BaB algorithms, each with its benefits, described in Sections 4.1 and 4.2. Finally, we combine the results to formulate an efficiently optimizable MILP instance in Section 5

4.1 Strong Bounding

Before going into the details, we briefly review the cross-executional bound refinement proposed in RACoon. For k-UAP, given any subset S of the k executions, RACoon can verify the absence of any common perturbation that works for **all** executions in S with cross-executional bound refinement. Let for all $i \in S$, $(\mathbf{L}_i(\boldsymbol{\alpha}_i), \mathbf{b}_i(\boldsymbol{\alpha}_i))$ denote the parametric linear approximations corresponding to the i-th execution. Then the optimal value $t^* = \max_{\boldsymbol{\alpha}_i, \lambda_i} -\epsilon \times \| \sum_{i \in S} \lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i)\|_1 + \sum_{i \in S} \lambda_i \times a_i(\boldsymbol{\alpha}_i) \geq 0$ proves absence of common perturbation $\boldsymbol{\delta}$ for S. Here, ϵ is the perturbation bound i.e. $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$, $a_i(\boldsymbol{\alpha}_i) = \mathbf{b}_i(\boldsymbol{\alpha}_i) + \mathbf{L}_i(\boldsymbol{\alpha}_i)^T \mathbf{x_i}$ and $\lambda_i \in [0, 1]$ with $\sum_{i \in S} \lambda_i = 1$ are the cross-executional parameters that relate linear approximations from different execution enabling joint optimization over $\boldsymbol{\alpha}_i$s. Next, we detail the first BaB method - strong bounding that combines cross-executional bounding with branching methods to verify the absence of common perturbation for any subset of $n = |S|$ executions.

Branching and bounding: For n executions, we construct a "product DNN" by duplicating the DNN n times, one for each execution. Formally, product DNN is a function $N^n : \mathbb{R}^{n_0 \times n} \to \mathbb{R}^{n_l \times n}$ with $N^n(\mathbf{x_1}, \ldots, \mathbf{x_n}) = [N(\mathbf{x_1}), \ldots, N(\mathbf{x_n})]^T$. At each branching step, we greedily select a subset of unbranched ReLU activations from the product DNN and branch on them, while using parametric linear relaxations for the rest. We adapt existing greedy branching heuristics, such as BaBSR Bunel et al. [2020b], for selecting the candidate ReLU activations. The heuristic computes a score for each unbranched ReLU activation in the product DNN, and we branch on the activations with the highest scores. Next, we detail the bounding method applied to each subproblem resulting from branching. Since the number of subproblems can be large, the bounding method needs to be fast yet capable of leveraging both branching constraints and cross-executional dependencies. However, the cross-executional bound refinement from RACoon can not handle branching constraints, while the bounding step from α, β-CROWN does not utilize dependencies across executions. Hence, we develop a three-step algorithm for obtaining the optimal value t^* with fast gradient descent-based methods. First, we replace these branching constraints by introducing dual variables $\boldsymbol{\beta}$, resulting in new parametric linear approximations $(\mathbf{L}_i(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i), b_i(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i))$ for each subproblem for all $i \in S$. Then for each subproblem, we introduce additional variables λ_i for each execution with constraints $\lambda_i \in [0, 1]$ and $\sum_{i \in S} \lambda_i = 1$. These λ_is relate linear approximations from different executions capturing cross-executional dependencies. This reduces finding t^* for each subproblem to the following optimization problem $t^* = \max_{\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i, \lambda_i} -\epsilon \times \| \sum_{i \in S} \lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i)\|_1 + \sum_{i \in S} \lambda_i \times a_i(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i)$. Here, $a_i(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i) = b_i(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i) + \mathbf{L}_i(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i)^T \mathbf{x_i}$. Finally, we apply projected gradient ascent to refine parameters $(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i, \lambda_i)$. The detailed derivation of the bounding step and the proof of correctness is in Appendix B. Precision gains of strong bounding over the baselines are in Section 6.2. Suppose, $\mathcal{F}(S)$ denotes the set of subproblems then Theorem 4.1 proves the absence of common perturbation for the subset S.

Theorem 4.1. *If* $\min_{\mathcal{F}(S)} \max_{\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i, \lambda_i} -\epsilon \times \| \sum_{i \in S} \lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i)\|_1 + \sum_{i \in S} \lambda_i \times a_i(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i) \geq 0$ *then executions in S do not have common perturbation* $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ *with* $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$.

Proof: The detailed proof in the Appendix B.

While strong bounding effectively combines cross-executional refinement with branching, it has the following drawbacks that led to the development of the 2nd BaB method. First, strong bounding branches over all executions simultaneously, which limits the number of branches explored per execution within a fixed timeout compared to branching on individual executions. For instance, if strong bounding solves m subproblems for n executions, then assuming each execution branched uniformly, each execution gets only $m^{\frac{1}{n}}$ subproblems. In contrast, given the same timeout, branching individually allows exploration $\frac{m}{n}$ subproblem per execution. Second, strong bounding only proves the absence of common perturbation, a relaxation of the k-UAP problem. To mitigate this, RACoon uses parameter refinement to obtain linear approximations and formulate a MILP, providing a more precise bound on k-UAP accuracy. However, for strong bounding, as the number of subproblems increases and each subproblem has a different linear approximation, formulating a MILP with each linear approximation is practically infeasible. Restricting the number of linear approximations can help accommodate MILP formulation by compromising on the method's strong bounding step.

4.2 Strong Branching

Unlike strong bounding, strong branching explores more branches by branching on each execution independently. Additionally, for each execution, we aim to keep the number of linear approximations small post-branching, ensuring the MILP instance using these approximations remains easy to optimize. To limit the number of linear approximations for each execution i, we fix a set of linear coefficients $\{\mathbf{L}_1, \dots, \mathbf{L}_m\}$ called "target coefficients" and for each $j \in [m]$, $\mathbf{L}_j \in \mathbb{R}^{n_0}$ compute valid lower bound b_j^* of the following optimization problem $\min_{\boldsymbol{\delta}} N(\mathbf{x_i} + \boldsymbol{\delta}) - \mathbf{L}_j^T(\mathbf{x_i} + \boldsymbol{\delta})$ with $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$ using BaB. In this case, for all $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ with $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$ the refined bias b_j^* and $\mathbf{L}_j$ remain a valid lower bound of $N(\mathbf{x_i} + \boldsymbol{\delta})$ i.e. $\mathbf{L}_j^T(\mathbf{x_i} + \boldsymbol{\delta}) + b_j^* \leq N(\mathbf{x_i} + \boldsymbol{\delta})$. Moreover, since we only refine the bias, the number of linear approximations remains the same as at the start of BaB, even after branching. Next, we describe how we utilize cross-execution dependencies while branching on each execution independently.

Selecting targets: We select target coefficients for each execution to facilitate relational verification. To select target coefficients, we greedily pick subsets of executions and run cross-executional refinement from RACoon without branching on each subset of executions. We describe the greedy selection strategy in Section 5. For each set of executions, we add the linear approximations obtained by cross-executional refinement to the corresponding executions' target sets. Cross-executional refinement ensures for each execution set the parameters are tailored for the relational verification.

Bounding and branching: Given a target coefficient $\mathbf{L}_t \in \mathbb{R}^{n_0}$, since finding the exact solution of $\min_{\boldsymbol{\delta}} N(\mathbf{x_i} + \boldsymbol{\delta}) - \mathbf{L}_t^T(\mathbf{x_i} + \boldsymbol{\delta})$ is computationally expensive, strong branching aims to obtain a tight mathematically correct lower bound on the difference $N(\mathbf{x_i}+\boldsymbol{\delta}) - \mathbf{L}_t^T(\mathbf{x_i}+\boldsymbol{\delta})$. For any subproblem, let $(\mathbf{L}(\boldsymbol{\alpha}, \boldsymbol{\beta}), b(\boldsymbol{\alpha}, \boldsymbol{\beta}))$ denote the parametric linear approximation. Then for this particular subproblem, for all $\boldsymbol{\alpha}, \boldsymbol{\beta}$, $\mathbf{L}(\boldsymbol{\alpha}, \boldsymbol{\beta})^T(x_i + \boldsymbol{\delta}) + b(\boldsymbol{\alpha}, \boldsymbol{\beta}) \leq N(\mathbf{x_i} + \boldsymbol{\delta})$ and subsequently:

$$\max_{\boldsymbol{\alpha}, \boldsymbol{\beta}} \min_{\|\boldsymbol{\delta}\|_\infty \leq \epsilon} (\mathbf{L}(\boldsymbol{\alpha}, \boldsymbol{\beta}) - \mathbf{L}_t)^T(\mathbf{x_i} + \boldsymbol{\delta}) + b(\boldsymbol{\alpha}, \boldsymbol{\beta}) \leq \min_{\|\boldsymbol{\delta}\|_\infty \leq \epsilon} N(\mathbf{x_i} + \boldsymbol{\delta}) - \mathbf{L}_t^T(\mathbf{x_i} + \boldsymbol{\delta}) \quad (1)$$

The optimal solution of the max-min problem in Eq. 1 provides a mathematically correct lower bound of $\min_{\boldsymbol{\delta}} N(\mathbf{x_i} + \boldsymbol{\delta}) - \mathbf{L}_t^T(\mathbf{x_i} + \boldsymbol{\delta})$ for each subproblem. However, it is hard to solve a max-min problem with scalable differentiable optimization techniques like gradient descent typically used for large DNNs considered in this paper. Instead, we compute a closed form of the inner minimization problem reducing the optimization instance to a more tractable maximization problem (Theorem 4.2).

Theorem 4.2. *For any $\boldsymbol{\alpha}, \boldsymbol{\beta}$, if $\mathbf{L}(\boldsymbol{\alpha}, \boldsymbol{\beta}) \in \mathbb{R}^{n_0}$ and $b(\boldsymbol{\alpha}, \boldsymbol{\beta}) \in \mathbb{R}$ then $\min_{\|\boldsymbol{\delta}\|_\infty \leq \epsilon}(\mathbf{L}(\boldsymbol{\alpha}, \boldsymbol{\beta}) - \mathbf{L}_t)^T(\mathbf{x} + \boldsymbol{\delta}) + b(\boldsymbol{\alpha}, \boldsymbol{\beta}) = -\epsilon \times \|\mathbf{L}(\boldsymbol{\alpha}, \boldsymbol{\beta}) - \mathbf{L}_t\|_1 + (\mathbf{L}(\boldsymbol{\alpha}, \boldsymbol{\beta}) - \mathbf{L}_t)^T\mathbf{x} + b(\boldsymbol{\alpha}, \boldsymbol{\beta})$.*

Proof: The proof is in Appendix C.
We apply a projected gradient ascent to optimize the maximization with the closed form obtained above (Appendix C.1). The proof of the correctness of the bounding method is in Appendix C. Note the proof of correctness does not necessitate the optimizer to find the global optimum. This is important since gradient ascent may not always converge to the global optimum. Since strong branching branch on each execution independently we reuse the branching strategy of α, β-CROWN.

5 RABBit

In this section, we detail the algorithm (Alog. 1) that combines the results from strong bounding and strong branching to formulate the MILP. Running strong bounding on all $2^k - 1$ non-empty subsets of k executions is impractical. Therefore, we use a greedy approach to select subsets of executions for strong bounding. Similarly, for strong branching, we greedily select the target linear coefficients. First, we describe both greedy strategies before moving on to the MILP formulation.

Elimination of individually verified executions: RABBit maintains a list of unverified indices and eliminates any executions that can be verified individually and does not consider them for subsequent steps (lines 3, 8, and 13 in Algo. 1). For instance, for k-UAP verification, we do not need to consider those executions that are proved to have no adversarial perturbation $\boldsymbol{\delta}$ such that $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$. Pruning individually verified executions improves the runtime without any compromise on the precision of the relational verifier (see Theorem B.1 Banerjee and Singh [2024]).
Greedy target coefficient selection: RABBit first runs RACoon which in turn executes an incomplete non-relation verifier α-CROWN Xu et al. [2021] eliminating the verified executions (line 8 in Algo. 1). Subsequently, for target selection, RABBit greedily picks the first k_t (hyperparameter) executions based on s_i the lower bound on $N(\mathbf{x_i} + \boldsymbol{\delta})$ as computed by α-CROWN, prioritizing executions with higher s_i (line 9). Intuitively, for unverified executions, s_i measures the maximum violation of the output specification $\psi^i(N(\mathbf{x_i} + \boldsymbol{\delta}))$ and thus leads to the natural choice of picking executions with smaller violations. For each selected execution i, we choose up to m target coefficients by iterating over all subsets $i \in S$ considered by RACoon, and selecting linear approximations corresponding to the top m subsets.The cross-executional lower bound t^* from RACoon decides the priority of each subset S. Subsets S with higher t^* indicate smaller violations and are more likely to be verified for the absence of a common perturbation, making them suitable for target selection.

Selection of subsets of executions for strong bounding: Thereafter, until timeout ζ, we run strong bounding on subsets of executions from individually unverified executions I. For each subset $S \subseteq I$, the cross-executional bound obtained by RACoon on S decides its priority. However, considering all non-empty subsets of I can be expensive. Instead, similar to strong branching, we first pick top-k_t executions (I_2) from I (Algo 1 line 19). We sort all non-empty subsets $S \subseteq I_2$ based on their priority and, in each iteration, run strong bounding on the highest-priority subset that has not been scheduled yet (Algo 1 line 22). Given a large timeout, RABBit would eventually select all subsets from I_2.

MILP Formulation: The MILP formulation uses both the refined biases from strong branching (line 11) and the subsets S of executions verified for the absence of common perturbation from strong bounding (line 22) to compute final verified UAP accuracy. RABBit MILP formulation involves three steps. First, we deduce linear constraints between the input and output of N for each unverified execution using linear approximations of N with refined bias obtained by strong branching. Secondly, we add constraints for each subset S verified for the absence of common perturbation with strong bounding. Then, similar to the current SOTA baseline Banerjee and Singh [2024], we encode the output specification Ψ as a MILP objective, introducing only $O(k)$ integer variables. Finally, we use an off-the-shelf MILP solver Gurobi Optimization, LLC [2018] to optimize the MILP.
Ψ encoding: First, we show the MILP objective $\mathbf{M}$ that encodes Ψ. We introduce binary variables $z_i \in \{0, 1\}$ for each individually unverified execution in I where for any perturbation $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ and $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$, $z_i = 1$ implies $\psi^i(N(x_i + \boldsymbol{\delta})) = True$. Then the finding the worst case UAP accuracy is equivalent to the following $\mathbf{M} = \frac{1}{k} \times \big((k - |I|) + min_{\|\delta\|_\infty \epsilon} \sum_{i \in I} z_i\big)$.

Algorithm 1 RABBit

```
1: Input: $N$, $(\Phi, \Psi)$, $k$, $k_t$, timeout $\zeta$
2: Output: $\mathbf{M}$.
3: $I \leftarrow \{\}$                      ▷ Unverified indices
4: $\mathcal{L} \leftarrow \{\}$                      ▷ Linear approximations
5: $C \leftarrow \{\}$                      ▷ Cross-verified executions
6: $\mathbf{s} \leftarrow \{\}$                      ▷ Lower bounds from $\alpha$-Crown
7: $\mathbf{M} \leftarrow 0$                      ▷ Initialize verified UAP accuracy
8: $(I, \mathcal{L}, C, \mathbf{s}) \leftarrow$ RACoon($N$, $(\Phi, \Psi)$, $k$)
9: $I_1 \leftarrow$ top-$k_t$ indices from $I$ based on $\mathbf{s}$
10: for $i \in I_1$ do
11:     $b_i^* \leftarrow$ StrongBranching($\phi^i, \psi^i, \mathcal{L}[i]$)
12:     if Verified($\phi^i, \psi^i, \mathcal{L}[i], b_i^*$) then
13:         $I \leftarrow I \setminus \{i\}$
14:     end if
15:     UpdateBias($\mathcal{L}[i], b_i^*$)
16:     $\mathcal{M} \leftarrow$MILP($\mathcal{L}$, $\Phi$, $\Psi$, $k$, $I$, $C$)
17:     $\mathbf{M} \leftarrow \max(\mathbf{M}(\Phi, \Psi), \text{Opt}(\mathcal{M}))$
18: end for
19: $I_2 \leftarrow$ top-$k_t$ indices from $I$ based on $\mathbf{s}$
20: while time() < $\zeta$ do
21:     $S \leftarrow$ Greedily select subset of $I_2$
22:     $t_S \leftarrow$ StrongBounding($S, \Phi, \Psi$)
23:     if $t_S \geq 0$ then
24:         $C \leftarrow$ Append($C, S$)
25:         $\mathcal{M} \leftarrow$MILP($\mathcal{L}$, $\Phi$, $\Psi$, $k$, $I$, $C$)
26:         $\mathbf{M} \leftarrow \max(\mathbf{M}, \text{Opt}(\mathcal{M}))$
27:     end if
28: end while
29: return $\mathbf{M}$
```

Constraints encoding: We add constraints from strong bounding, strong branching, and from the linear approximation obtained from the call to RACoon (Algo. 1 line 8). Suppose for any subset $S \subseteq I$, strong bounding verifies the absence of common perturbation. Then for all $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ and $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$ at least one of the executions from S will always satisfy the corresponding output specification. Hence, for every such S we add the constraint: $\sum_{i \in S} z_i \geq 1$. Now, let for any $i \in I$, $\{(\mathbf{L}_i^1, b_i^1), \ldots, (\mathbf{L}_i^m, b_i^m)\}$ denote set of linear approximation with b_i^m either coming from RACoon or from strong branching. Then we add the following constraints $z_i \geq z'_i$, $z'_i = (o_i \geq 0)$, $o_i \geq \mathbf{L}_i^{jT}(\mathbf{x_i} + \boldsymbol{\delta}) + b_i^j$ where $o_i \in \mathbb{R}$, z'_i are newly introduced real and integer variables respectively.

Limitations: Although RABBit outperforms SOTA verifiers in relational verification, like all deterministic verifiers, whether relational or non-relational (including ours), do not scale to deep neural networks (DNNs) trained on very large datasets such as ImageNet. RABBit is sound but incomplete, meaning it may not be able to prove certain relational properties even if they are true. Note that all complete non-relational verifiers are also incomplete for relational properties since they do not track any dependencies between executions.

6 Experimental Evaluation

We evaluate the effectiveness of RABBit on multiple relational properties, DNNs, and datasets. In our evaluation, we compare RABBit against SOTA baselines, including non-relational verifiers CROWN Zhang et al. [2018], α-CROWN Xu et al. [2021], α, β-CROWN Wang et al. [2021b], as well as relational verifiers I/O Formulation Zeng et al. [2023] and RACoon. Additionally, we show that: a) given the same time, RABBit always outperforms the SOTA BaB-based non-relational verifier α, β-CROWN; b) strong bounding computes a tighter bound on t^* than α, β-CROWN; and c) we provide an ablation study on ϵ, k, and the hyperparameter k_t used by RABBit.

6.1 Experiment Setup

Networks. We use standard convolutional and residual architectures, such as ConvSmall and ConvBig, which are used to evaluate both SOTA relational Wang et al. [2021b] and non-relational verifiers Banerjee and Singh [2024] (see Table 1). We provide the details of the DNN architectures in the Appendix D.1. We use networks trained using both standard training methods and robust training strategies, such as DiffAI Mirman et al. [2018], SABR Mueller et al. [2023], and CITRUS Xu and Singh [2024]. Our experiments utilize publicly available pre-trained DNNs sourced from the CROWN repository Zhang et al. [2020], α, β-CROWN repository Wang et al. [2021b], and ERAN repository Singh et al. [2019b]. The clean accuracies of these networks are reported in Appendix D.2.

Implementation details and hyperparameters. We implemented our method in Python with Pytorch V1.11 on top of SOTA complete non-relational verifier α, β-CROWN Wang et al. [2021b]. We used Gurobi V11.0 as the off-the-shelf MILP solver. For both strong bounding and strong branching, we use Adam Kingma and Ba [2014] for parameter learning and run it for 20 iterations

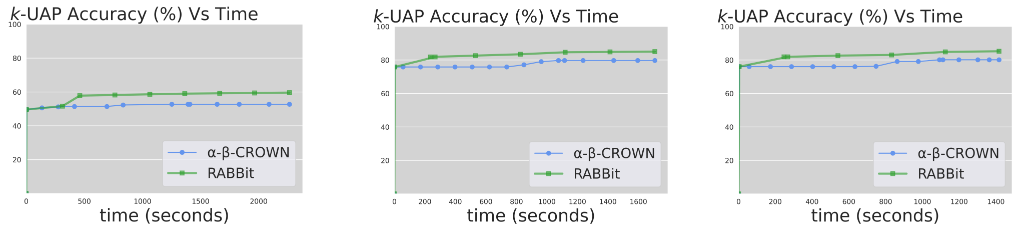
Table 1: RABBit Efficacy Analysis for Worst-Case UAP Accuracy

Dataset	Network Structure	Training Method	Perturbation Bound (ϵ)	CROWN	$\alpha-$CROWN	$\alpha, \beta-$CROWN	I/O	RACoon	Strong Bounding	Strong Branching	RABBit
CIFAR10	ConvSmall	Standard	1/255	44.8	45.4	59.8	45.4	45.4	60.0 (+0.2)	60.6 (+0.8)	62.4 (+2.6)
	ConvSmall	DiffAI	5/255	44.4	49.6	53.6	50.4	51.6	59.0 (+5.4)	59.0 (+5.4)	59.8 (+6.2)
	ConvSmall	SABR	2/255	75.2	75.8	78.4	76.8	78.2	83.0 (+4.6)	83.8 (+5.4)	84.0 (+5.6)
	ConvSmall	CITRUS	2/255	74.8	76.0	79.0	77.0	78.8	82.8 (+3.8)	83.2 (+4.2)	83.6 (+4.6)
	ConvBig	DiffAI	2/255	46.6	51.8	57.2	53.2	54.8	59.8 (+2.6)	60.0 (+2.8)	60.4 (+3.2)
	ResNet-2B	Standard	1/255	52.6	52.6	56.0	53.6	55.0	56.2 (+0.6)	56.2 (+0.6)	57.0 (+1.0)
MNIST	ConvSmall	Standard	0.10	7.8	9.8	32.8	16.0	18.0	35.4 (+2.6)	34.8 (+2.0)	36.2 (+3.4)
	ConvSmall	DiffAI	0.13	51.8	57.0	72.8	57.2	58.4	74.6 (+1.8)	74.2 (+1.4)	75.2 (+2.4)
	ConvSmall	SABR	0.15	27.0	38.0	50.4	42.2	45.8	51.4 (+0.8)	51.4 (+0.8)	52.2 (+1.8)
	ConvSmall	CITRUS	0.15	28.8	41.6	59.4	41.6	44.6	60.6 (+1.2)	60.0 (+0.6)	61.6 (+2.2)
	ConvBig	DiffAI	0.2	81.4	86.6	89.6	86.6	87.0	90.6 (+1.0)	90.6 (+1.0)	91.4 (+1.8)

on each subproblem. We set the value of $k_t = 10$ for CIFAR-10 and $k_t = 20$ for MNIST networks respectively. We use a single NVIDIA A100-PCI GPU with 40 GB RAM for bound refinement and an Intel(R) Xeon(R) Silver 4214R CPU @ 2.40GHz with 64 GB RAM for MILP optimization. For any relational property with k executions, we give an overall timeout of k minutes (averaging 1 minute/execution) to RABBit and all baselines. Each MILP instance gets a timeout of 5 minutes.

6.2 Experimental Results

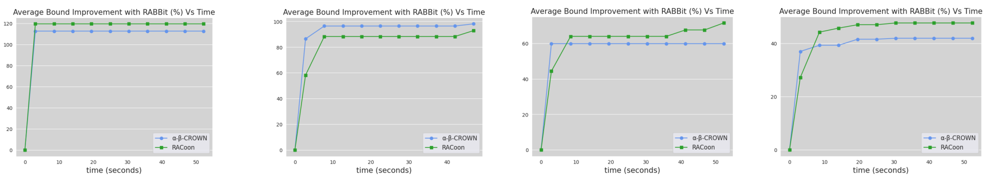
Effectiveness of RABBit: Table 1 compares the results of RABBit to all baselines across different datasets (column 1) and DNN architectures (column 2) trained with various methods (column 3), with ϵ values defining the L_∞ bound of $\boldsymbol{\delta}$ in column 4. For each DNN and ϵ, we run RABBit and all the baselines on 10 relational properties each defined with $k = 50$ randomly selected inputs, and report the worst-case UAP accuracy averaged over the 10 properties. Note that for each DNN, we exclude inputs misclassified by the DNN. We compare the performance of RABBit against SOTA relational and complete non-relational verifiers as well as against strong bounding and strong branching.
The results in Table 1 demonstrate that strong bounding, strong branching, and RABBit all outperform the existing SOTA verifiers on all DNNs and ϵ. Notably, RABBit gains up to $+6.2\%$ and up to $+3.4\%$ improvement in the worst-case UAP accuracy (averaged over 10 runs) for CIFAR10 and MNIST DNNs, respectively. RABBit also efficiently scales to the largest verifiable DNN architectures such as ResNet and ConvBig, conferring up to +3.2% improvement in worst-case UAP accuracy. In some cases, strong bounding outperforms strong branching, while in others, strong branching outperforms strong bounding, highlighting the importance of both methods. RABBit combines the strengths of both strong branching and strong bounding, producing the best results overall.



(a) DiffAI (CIFAR10) (b) SABR (CIFAR10) (c) CITRUS (CIFAR10)

Figure 1: Average Worst Case k-UAP accuracy vs Time for ConvSmall CIFAR10 DNNs.

Time vs UAP Accuracy Analysis: Fig. 1 shows timewise the worst-case UAP accuracy (averaged over 10 runs) for different ConvSmall CIFAR10 networks with $k = 50$ on ϵ values from Table 1. Note that RABBit invokes RACoon, which in turn calls α-CROWN and eliminates verified executions (Line 7 in Algorithm 1). Hence, for a fair comparison, we also run α-CROWN first for α, β-CROWN and then run α, β-CROWN only on the unverified indices. For all DNNs, RABBit consistently outperforms the SOTA BaB-based non-relational verifier α, β-CROWN at all timestamps. This confirms that the improved precision shown in Table 1 is not dependent on the specific timeout value.
Evaluating Bound Improvement: In Fig 2, we present a timewise analysis of the improvement in t^* with strong bounding over α, β-CROWN and RACoon. For this experiment, we use DiffAI and CITRUS ConvSmall networks with epsilon values from Table 1. For each network and ϵ, we select 30 executions at random and compute the percentage improvement in t^* with strong bounding over RACoon and α, β-CROWN.We also report the average improvement and 95% confidence intervals for all cases in Table 4 in Appendix E. The results demonstrate that the t^* with strong bounding is

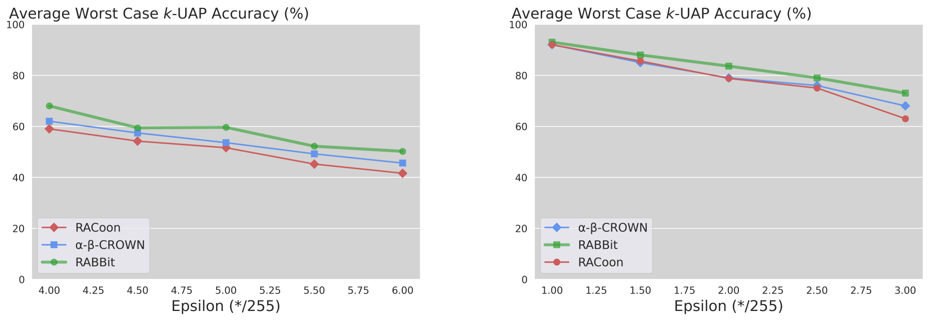


(a) DiffAI (CIFAR10) (b) CITRUS (CIFAR10) (c) DiffAI (MNIST) (d) CITRUS (MNIST)

Figure 2: Timewise Analysis of Average % Improvement in t^* with Strong Bounding

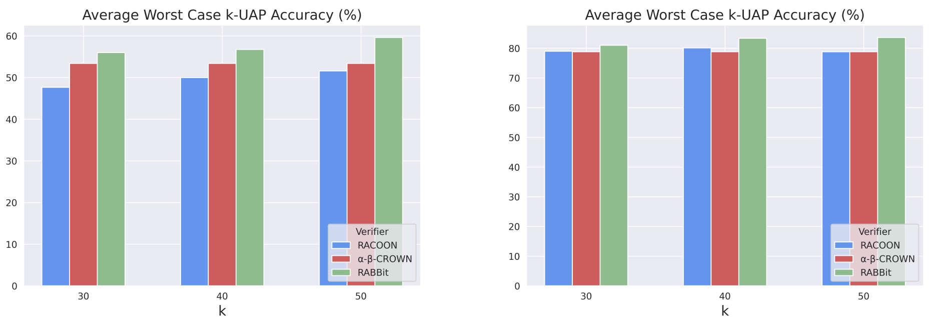
significantly tighter compared to the bounds from the SOTA verifiers α, β-CROWN and RACoon at all timestamps. Furthermore, strong bounding improves t^* on average by up to 108.7% for CIFAR10 networks and 57.7% for MNIST networks. These results highlight the importance of leveraging dependencies across executions during both branching and bounding to improve precision.

Different ϵ and k values: Fig. 3 shows the results of RACoon, α, β-CROWN, and RABBit for k-UAP verification of CIFAR10 ConvSmall DNNs for 5 different ϵ values and $k = 50$. We also report ϵ ablation results for MNIST DNNs in Appendix G.1.RABBit outperforms RACoon and α, β-CROWN for all evaluated ϵ values, notably improving the worst case k-UAP accuracy by up to 6.2%. Similarly, we analyze the performance of RACoon, α, β-CROWN, and RABBit for k-UAP verification of CIFAR10 ConvSmall DNNs with different k values. Results for MNIST DNNs are presented in Appendix G.2. As presented in Fig. 4, for all k values, RABBit is more precise than both baselines. Expectedly, the worst-case k-UAP accuracy for relational verifiers is higher with larger k values as it is easier to prove the absence of a common perturbation with larger k.



(a) DiffAI (CIFAR10) (b) CITRUS (CIFAR10)

Figure 3: Average Worst Case k-UAP accuracy vs ϵ for CIFAR10 DNNs.



(a) DiffAI (CIFAR10) (b) CITRUS (CIFAR10)

Figure 4: Average Worst Case k-UAP accuracy for different k values for CIFAR10 ConvSmall DNNs.

7 Conclusion

We present RABBit, a general framework for improving the precision of relational verification of DNNs through BaB methods specifically designed to utilize dependencies across executions. Our experiments, on various DNN architectures, and training methods demonstrate that RABBit significantly outperforms both SOTA relational and non-relational verifiers for relational properties. Although we focus on the worst-case UAP accuracy RABBit can be extended to properties involving different DNNs, such as local equivalence of DNN pairs Paulsen et al. [2020] or properties defined over an ensemble of DNNs.

References

Filippo Amato, Alberto López, Eladia María Peña-Méndez, Petr Vaňhara, Aleš Hampl, and Josef Havel. Artificial neural networks in medical diagnosis. *Journal of Applied Biomedicine*, 11(2), 2013.

Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 2020.

Stanley Bak, Hoang-Dung Tran, Kerianne Hobbs, and Taylor T. Johnson. Improved geometric path enumeration for verifying relu neural networks. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I*, volume 12224 of *Lecture Notes in Computer Science*, pages 66–96. Springer, 2020. doi: 10.1007/978-3-030-53288-8_4. URL `https://doi.org/10.1007/978-3-030-53288-8_4`.

Debangshu Banerjee and Gagandeep Singh. Relational dnn verification with cross executional bound refinement, 2024.

Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Pushmeet Kohli, P Torr, and P Mudigonda. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21 (2020), 2020a.

Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, M. Pawan Kumar, Jingyue Lu, and Pushmeet Kohli. Branch and bound for piecewise linear neural network verification. *J. Mach. Learn. Res.*, 21(1), jan 2020b. ISSN 1532-4435.

Rudy R Bunel, Oliver Hinder, Srinadh Bhojanapalli, and Krishnamurthy Dvijotham. An efficient non-convex reformulation of stagewise convex optimization problems. *Advances in Neural Information Processing Systems*, 33, 2020c.

Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1310–1320. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/cohen19c.html`.

Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, 2017.

Claudio Ferrari, Mark Niklas Mueller, Nikola Jovanović, and Martin Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=l_amHf1oaK`.

Aymeric Fromherz, Klas Leino, Matt Fredrikson, Bryan Parno, and Corina Pasareanu. Fast geometric projections for local robustness certification. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=zWy1uxjDdZJ`.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, 2018.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2018.

Haitham Khedr and Yasser Shoukry. Certifair: A framework for certified global fairness of neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(7):8237–8245, Jun. 2023.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Juncheng Li, Shuhui Qu, Xinjian Li, Joseph Szurley, J. Zico Kolter, and Florian Metze. Adversarial music: Real world audio adversary against wake-word detection system. In *Proc. Neural Information Processing Systems (NeurIPS)*, pages 11908–11918, 2019a.

Juncheng Li, Frank R. Schmidt, and J. Zico Kolter. Adversarial camera stickers: A physical camera-based attack on deep learning systems. In *Proc. International Conference on Machine Learning, ICML*, volume 97, pages 3896–3904, 2019b.

Linyi Li, Jiawei Zhang, Tao Xie, and Bo Li. Double sampling randomized smoothing. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 13163–13208. PMLR, 17–23 Jul 2022. URL `https://proceedings.mlr.press/v162/li22aa.html`.

Zikun Liu, Changming Xu, Emerson Sie, Gagandeep Singh, and Deepak Vasisht. Exploring practical vulnerabilities of machine learning-based wireless systems. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023*, pages 1801–1817. USENIX Association, 2023.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJzIBfZAb`.

Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3578–3586. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/mirman18b.html`.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.

Mark Niklas Mueller, Franziska Eckert, Marc Fischer, and Martin Vechev. Certified training: Small boxes are all you need. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=7oFuxtJtUMH`.

Alessandro De Palma, Harkirat S. Behl, Rudy R. Bunel, Philip H. S. Torr, and M. Pawan Kumar. Scaling the convex barrier with active sets. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

Brandon Paulsen, Jingbo Wang, and Chao Wang. Reludiff: Differential verification of deep neural networks. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, page 714–726, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371216. doi: 10.1145/3377811.3380337. URL `https://doi.org/10.1145/3377811.3380337`.

Brandon Paulsen, Jingbo Wang, Jiawei Wang, and Chao Wang. Neurodiff: Scalable differential verification of neural networks using fine-grained approximation. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, ASE '20, page 784–796, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450367684. doi: 10.1145/3324884.3416560. URL `https://doi.org/10.1145/3324884.3416560`.

Yannik Potdevin, Dirk Nowotka, and Vijay Ganesh. An empirical investigation of randomized defenses against adversarial attacks. *arXiv preprint arXiv:1909.05580*, 2019.

Zhouxing Shi, Qirui Jin, J Zico Kolter, Suman Jana, Cho-Jui Hsieh, and Huan Zhang. Formal verification for neural networks with general nonlinearities via branch-and-bound, 2024. URL `https://openreview.net/forum?id=ivokwVKY4o`.

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. *Advances in Neural Information Processing Systems*, 31, 2018.

Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. Beyond the single neuron convex barrier for neural network certification. In *Advances in Neural Information Processing Systems*, 2019a.

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

Matthew Sotoudeh and Aditya V Thakur. Abstract neural networks. In *Static Analysis: 27th International Symposium, SAS 2020, Virtual Event, November 18–20, 2020, Proceedings 27*, pages 65–88. Springer, 2020.

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

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. *arXiv preprint arXiv:2103.06624*, 2021a.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021b. URL https://openreview.net/forum?id=ahYIlRBeCFw.

Haoze Wu, Teruhiro Tagomori, Alexander Robey, Fengjun Yang, Nikolai Matni, George Pappas, Hamed Hassani, Corina Pasareanu, and Clark Barrett. Toward certified robustness against real-world distribution shifts. In *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pages 537–553. IEEE, 2023a.

Haoze Wu, Teruhiro Tagomori, Alexander Robey, Fengjun Yang, Nikolai Matni, George Pappas, Hamed Hassani, Corina Pasareanu, and Clark Barrett. Toward certified robustness against real-world distribution shifts. In *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pages 537–553. IEEE, 2023b.

Chulin Xie, Minghao Chen, Pin-Yu Chen, and Bo Li. Crfl: Certifiably robust federated learning against backdoor attacks. In *International Conference on Machine Learning*, pages 11372–11382. PMLR, 2021.

Changming Xu and Gagandeep Singh. Cross-input certified training for universal perturbations, 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 *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=nVZtXBI6LNn.

Yi Zeng, Zhouxing Shi, Ming Jin, Feiyang Kang, Lingjuan Lyu, Cho-Jui Hsieh, and Ruoxi Jia. Towards robustness certification against universal perturbations. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=7GEvPKxjtt.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *Advances in neural information processing systems*, 31, 2018.

Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. In *Proc. International Conference on Learning Representations (ICLR)*, 2020.

Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. General cutting planes for bound-propagation-based neural network verification. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022a. URL `https://openreview.net/forum?id=5haAJAcofjc`.

Yuhao Zhang, Aws Albarghouthi, and Loris D'Antoni. Bagflip: A certified defense against data poisoning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022b. URL `https://openreview.net/forum?id=ZidkM5b92G`.

A Formal encoding of relational properties

A.1 k-UAP verification

Given a set of k points $\mathbf{X} = \{\mathbf{x_1}, ..., \mathbf{x_k}\}$ where for all $i \in [k]$, $\mathbf{x_i} \in \mathbb{R}^{n_0}$ and $\epsilon \in \mathbb{R}$ we can first define individual input constraints used to define L_∞ input region for each execution $\forall i \in [k].\phi^i_{in}(\mathbf{x_i^*}) = \|\mathbf{x_i^*} - \mathbf{x_i}\|_\infty \leq \epsilon$. We define $\Phi^\delta(\mathbf{x_1^*}, \ldots, \mathbf{x_k^*})$ as follows:

$$\Phi^\delta(\mathbf{x_1^*}, \ldots, \mathbf{x_k^*}) = \bigwedge_{(i,j \in [k]) \wedge (i<j)} (\mathbf{x_i^*} - \mathbf{x_j^*} = \mathbf{x_i} - \mathbf{x_j}) \tag{2}$$

Then, we have the input specification as $\Phi(\mathbf{x_1^*}, \ldots, \mathbf{x_k^*}) = \bigwedge_{i=1}^k \phi^i_{in}(\mathbf{x_i^*}) \wedge \Phi^\delta(\mathbf{x_1^*}, \ldots, \mathbf{x_k^*})$.

Next, we define $\Psi(\mathbf{x_1^*}, \ldots, \mathbf{x_k^*})$ as conjunction of k clauses each defined by $\psi^i(\mathbf{y_i})$ where $\mathbf{y_i} = N(\mathbf{x_i^*})$. Now we define $\psi^i(\mathbf{y_i}) = \bigwedge_{j=1}^{n_l}(\mathbf{c_{i,j}}^T\mathbf{y_i} \geq 0)$ where $\mathbf{c_{i,j}} \in \mathbb{R}^{n_l}$ is defined as follows

$$\forall a \in [n_l].c_{i,j,a} = \begin{cases} 1 & \text{if } a \neq j \text{ and } a \text{ is the correct label for } \mathbf{y_i} \\ -1 & \text{if } a = j \text{ and } a \text{ is not the correct label for } \mathbf{y_i} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

In this case, the tuple of inputs $(\mathbf{x_1^*}, \ldots, \mathbf{x_k^*})$ satisfies the input specification $\Phi(\mathbf{x_1^*}, \ldots, \mathbf{x_k^*})$ iff for all $i \in [k]$, $\mathbf{x_i^*} = \mathbf{x_i} + \boldsymbol{\delta}$ where $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ and $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$. Hence, the relational property (Φ, Ψ) defined above verifies whether there is an adversarial perturbation $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ with $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$ that can misclassify **all** k inputs. Next, we show the formulation for the worst-case UAP accuracy of the k-UAP verification problem as described in section 3. Let, for any $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ and $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$, $\mu(\delta)$ denotes the number of clauses (ψ^i) in Ψ that are satisfied. Then $\mu(\delta)$ is defined as follows

$$z_i(\boldsymbol{\delta}) = \begin{cases} 1 & \psi^i(N(\mathbf{x_i} + \boldsymbol{\delta})) \text{ is } True \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$\mu(\boldsymbol{\delta}) = \sum_{i=1}^k z_i(\boldsymbol{\delta}) \tag{5}$$

Since $\psi^i(N(\mathbf{x_i} + \boldsymbol{\delta}))$ is $True$ iff the perturbed input $\mathbf{x_i} + \boldsymbol{\delta}$ is correctly classified by N, for any $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ and $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$, $\mu(\boldsymbol{\delta})$ captures the number of correct classifications over the set of perturbed inputs $\{\mathbf{x_1} + \boldsymbol{\delta}, \ldots, \mathbf{x_k} + \boldsymbol{\delta}\}$. The worst-case k-UAP accuracy $\mathbf{M}_0(\Phi, \Psi)$ for (Φ, Ψ) is as follows

$$\mathbf{M}_0(\Phi, \Psi) = \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0},\ \|\boldsymbol{\delta}\| \leq \epsilon} \mu(\boldsymbol{\delta}) \tag{6}$$

B Details of strong bounding

We first show that given fixed linear approximations $\{(\mathbf{L}_1, b_1), \ldots (\mathbf{L}_n, b_n)\}$ corresponding to n executions of N if the optimal value t^* of the following linear program ≥ 0 then the n executions do not have a common peturbation.

$$\begin{aligned} &min\ \ t \quad \text{s.t. } \|\boldsymbol{\delta}\|_\infty \leq \epsilon \\ &\mathbf{L_i}^T(\mathbf{x_i} + \boldsymbol{\delta}) + b_i \leq t \quad \forall i \in [n] \end{aligned} \tag{7}$$

Now in the first step, we compute the Lagrangian dual of the linear program from Eq. 7. The Lagrangian Dual is as follows where for all $i \in [n]$, $\lambda_i \geq 0$ are Lagrange multipliers.

$$\max_{0 \leq \lambda_i} \min_{t \in \mathbb{R}, \|\boldsymbol{\delta}\|_\infty \leq \epsilon} (1 - \sum_{i=1}^n \lambda_i) \times t + \sum_{i=1}^n \lambda_i \times \left(\mathbf{L}_i^T(\mathbf{x_i} + \boldsymbol{\delta}) + b_i\right)$$

We set the coefficient of the unbounded variable t to 0 to avoid cases where $\min_{t \in \mathbb{R}, \|\boldsymbol{\delta}\|_\infty \leq \epsilon} (1 - \sum_{i=1}^n \lambda_i) \times t + \sum_{i=1}^n \lambda_i \times \left(\mathbf{L}_i^T(\mathbf{x_i} + \boldsymbol{\delta}) + b_i\right) = -\infty$. This leads to the following Lagrangian Dual form

$$\max_{0 \leq \lambda_i} \min_{\|\boldsymbol{\delta}\|_\infty \leq \epsilon} \sum_{i=1}^n \lambda_i \times \left(\mathbf{L}_i^T(\mathbf{x_i} + \boldsymbol{\delta}) + b_i\right) \quad \text{where} \sum_{i=1}^n \lambda_i = 1$$

Now for every subproblem, replacing the branching constraints with $\boldsymbol{\beta}$ dual variables results in the parametric linear approximations of N specified by $(\mathbf{L}_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i), b_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i))$ for each execution $i \in [n]$. Then the Lagrangian Dual with the parametric linear approximations $\{(\mathbf{L}_1(\boldsymbol{\alpha}_1,\boldsymbol{\beta}_1), b_1(\boldsymbol{\alpha}_1,\boldsymbol{\beta}_1)), \dots, (\mathbf{L}_n(\boldsymbol{\alpha}_n,\boldsymbol{\beta}_n), b_n(\boldsymbol{\alpha}_n,\boldsymbol{\beta}_n))\}$ is as follows

$$\max_{0\le\lambda_i} \min_{\|\boldsymbol{\delta}\|_\infty\le\epsilon} \sum_{i=1}^{n} \lambda_i \times \left(\mathbf{L}_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)^T(\mathbf{x_i}+\boldsymbol{\delta}) + b_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)\right) \quad \text{where} \sum_{i=1}^{n}\lambda_i = 1$$

Theorem 4.1. *If* $\min_{\mathcal{F}(S)} \max_{\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i,\lambda_i} -\epsilon \times \|\sum_{i\in S}\lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)\|_1 + \sum_{i\in S}\lambda_i \times a_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i) \ge 0$ *then executions in* S *do not have common perturbation* $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ *with* $\|\boldsymbol{\delta}\|_\infty \le \epsilon$.

Proof. First, we show that $\min\limits_{\|\boldsymbol{\delta}\|_\infty\le\epsilon} \sum_{i=1}^{n}\lambda_i \times \left(\mathbf{L}_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)^T(\mathbf{x_i}+\boldsymbol{\delta}) + b_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)\right) = -\epsilon \times \|\sum_{i=1}^{n}\lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)\|_1 + \sum_{i=1}^{n}\lambda_i \times a_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)$.

$$\begin{aligned}
&\min_{\|\boldsymbol{\delta}\|_\infty\le\epsilon} \sum_{i=1}^{n} \lambda_i \times \left(\mathbf{L}_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)^T(\mathbf{x_i}+\boldsymbol{\delta}) + b_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)\right)\\
&= \min_{\|\boldsymbol{\delta}\|_\infty\le\epsilon} \sum_{i=1}^{n} \lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)^T(\boldsymbol{\delta}) + \sum_{i=1}^{n}\lambda_i \times \left(b_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i) + \mathbf{L}_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)^T\mathbf{x_i}\right)\\
&= \sum_{i=1}^{n}\lambda_i \times a_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i) + \min_{\|\boldsymbol{\delta}\|_\infty\le\epsilon}\sum_{i=1}^{n}\lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)^T(\boldsymbol{\delta})\\
&= \sum_{i=1}^{n}\lambda_i \times a_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i) - \epsilon \times \|\sum_{i=1}^{n}\lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)\|_1 \quad \text{Using Hölder's Inequality} \qquad (8)
\end{aligned}$$

For fixed $\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i$, the optimal solution of the LP in Eq. 7 and subsequently of the Lagrangian gives us

$$\begin{aligned}
&\max_{0\le\lambda_i}\min_{\|\boldsymbol{\delta}\|_\infty\le\epsilon} \sum_{i=1}^{n} \lambda_i \times \left(\mathbf{L}_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)^T(\mathbf{x_i}+\boldsymbol{\delta}) + b_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)\right)\\
&= \min_{\|\boldsymbol{\delta}\|_\infty\le\epsilon}\max_{1\le i\le n} \left(\mathbf{L}_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)^T(\mathbf{x_i}+\boldsymbol{\delta}) + b_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)\right) \quad \text{provided} \sum_{i=1}^{n}\lambda_i = 1 \qquad (9)
\end{aligned}$$

For each subproblem, for all $\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i$

$$\min_{\|\boldsymbol{\delta}\|_\infty\le\epsilon}\max_{1\le i\le n} \mathbf{c_i}^T N(\mathbf{x_i}+\boldsymbol{\delta}) \ge \min_{\|\boldsymbol{\delta}\|_\infty\le\epsilon}\max_{1\le i\le n}\left(\mathbf{L}_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)^T(\mathbf{x_i}+\boldsymbol{\delta}) + b_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)\right)$$

Hence,

$$\begin{aligned}
&\min_{\|\boldsymbol{\delta}\|_\infty\le\epsilon}\max_{1\le i\le n} \mathbf{c_i}^T N(\mathbf{x_i}+\boldsymbol{\delta})\\
&\ge \max_{\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i}\min_{\|\boldsymbol{\delta}\|_\infty\le\epsilon}\max_{1\le i\le n}\left(\mathbf{L}_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)^T(\mathbf{x_i}+\boldsymbol{\delta}) + b_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)\right)\\
&\ge \max_{\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i}\max_{0\le\lambda_i}\min_{\|\boldsymbol{\delta}\|_\infty\le\epsilon}\sum_{i=1}^{n}\lambda_i \times \left(\mathbf{L}_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)^T(\mathbf{x_i}+\boldsymbol{\delta}) + b_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)\right) \quad \text{where} \sum_{i=1}^{n}\lambda_i = 1 \text{ from Eq. 9}\\
&\ge \max_{\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i,0\le\lambda_i}\sum_{i=1}^{n}\lambda_i \times a_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i) - \epsilon \times \|\sum_{i=1}^{n}\lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)\|_1 \quad \text{From Eq. 8} \qquad (10)
\end{aligned}$$

Finally, if $\min_{\mathcal{F}(S)} \max_{\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i,\lambda_i} -\epsilon \times \|\sum_{i\in S}\lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i)\|_1 + \sum_{i\in S}\lambda_i \times a_i(\boldsymbol{\alpha}_i,\boldsymbol{\beta}_i) \ge 0$ then,

$$\min_{\|\boldsymbol{\delta}\|_\infty\le\epsilon}\max_{1\le i\le n} \mathbf{c_i}^T N(\mathbf{x_i}+\boldsymbol{\delta}) \ge 0 \quad \text{using Eq. 10}$$

Since, $\min\limits_{\|\boldsymbol{\delta}\|_\infty\le\epsilon}\max_{1\le i\le n} \mathbf{c_i}^T N(\mathbf{x_i}+\boldsymbol{\delta}) \ge 0$, $\bigvee_{i=1}^{n}\psi^i(N(\mathbf{x_i}+\boldsymbol{\delta}))$ holds for all $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ and $\|\boldsymbol{\delta}\|_\infty \le \epsilon$ i.e. there does not exist any common perturbation. □

C Details of strong branching

Theorem 4.2. *For any $\boldsymbol{\alpha}, \boldsymbol{\beta}$, if* $\mathbf{L}(\boldsymbol{\alpha},\boldsymbol{\beta}) \in \mathbb{R}^{n_0}$ *and* $b(\boldsymbol{\alpha},\boldsymbol{\beta}) \in \mathbb{R}$ *then* $\min_{\|\boldsymbol{\delta}\|_\infty \le \epsilon}(\mathbf{L}(\boldsymbol{\alpha},\boldsymbol{\beta}) - \mathbf{L}_t)^T(\mathbf{x}+\boldsymbol{\delta}) + b(\boldsymbol{\alpha},\boldsymbol{\beta}) = -\epsilon \times \|\mathbf{L}(\boldsymbol{\alpha},\boldsymbol{\beta}) - \mathbf{L}_t\|_1 + (\mathbf{L}(\boldsymbol{\alpha},\boldsymbol{\beta}) - \mathbf{L}_t)^T\mathbf{x} + b(\boldsymbol{\alpha},\boldsymbol{\beta})$.

Proof.

$$\begin{aligned}
&\min_{\|\boldsymbol{\delta}\|_\infty \le \epsilon}(\mathbf{L}(\boldsymbol{\alpha},\boldsymbol{\beta}) - \mathbf{L}_t)^T(\mathbf{x}+\boldsymbol{\delta}) + b(\boldsymbol{\alpha},\boldsymbol{\beta}) \\
&= \min_{\|\boldsymbol{\delta}\|_\infty \le \epsilon}(\mathbf{L}(\boldsymbol{\alpha},\boldsymbol{\beta}) - \mathbf{L}_t)^T\boldsymbol{\delta} + b(\boldsymbol{\alpha},\boldsymbol{\beta}) + (\mathbf{L}(\boldsymbol{\alpha},\boldsymbol{\beta}) - \mathbf{L}_t)^T\mathbf{x} \\
&= b(\boldsymbol{\alpha},\boldsymbol{\beta}) + (\mathbf{L}(\boldsymbol{\alpha},\boldsymbol{\beta}) - \mathbf{L}_t)^T\mathbf{x} + \min_{\|\boldsymbol{\delta}\|_\infty \le \epsilon}(\mathbf{L}(\boldsymbol{\alpha},\boldsymbol{\beta}) - \mathbf{L}_t)^T\boldsymbol{\delta} \\
&= b(\boldsymbol{\alpha},\boldsymbol{\beta}) + (\mathbf{L}(\boldsymbol{\alpha},\boldsymbol{\beta}) - \mathbf{L}_t)^T\mathbf{x} - \epsilon \times \|(\mathbf{L}(\boldsymbol{\alpha},\boldsymbol{\beta}) - \mathbf{L}_t)\|_1 \quad \text{Using Hölder's Inequality}
\end{aligned}$$

□

C.1 Projected gradient descent

For each $\boldsymbol{\alpha_i}, \boldsymbol{\beta}_i$, after each step of gradient ascent (for maximization problem), we clip $\boldsymbol{\alpha_i}, \boldsymbol{\beta}_i$ values to the corresponding ranges $[\boldsymbol{l_i^\alpha}, \boldsymbol{u_i^\alpha}]$ $[\boldsymbol{l_i^\beta}, \boldsymbol{u_i^\beta}]$ respectively. This is similar to the approach used in the SOTA non-relational bound refinement α, β-CROWN Wang et al. [2021b]. Since $\lambda_i \in [0,1]$ and $\sum_{i=1}^k \lambda_i = 1$ we replace $\lambda_i = \frac{sigmoid(x_i)}{\sum_{i=1}^k sigmoid(x_i)}$ where $x_i \in \mathbb{R}$. For any values of $(x_1, \dots, x_k) \in \mathbb{R}^k$ the corresponding $(\lambda_1, \dots, \lambda_k)$ satisfy $\lambda_i \in [0,1]$ and $\sum_{i=1}^k \lambda_i = 1$. We then apply gradient ascent (for maximization problem) on $(x_1, \dots, x_k)$ without any constraints.

D DNN Architectures

D.1 DNN Architectures:

Table 2: DNN Architecture Details

Dataset	Model	Type	Train	# Layers	# Params
MNIST	ConvSmall	Conv	Standard	4	80k
	ConvSmall	Conv	DiffAI	4	80k
	ConvSmall	Conv	SABR	4	80k
	ConvSmall	Conv	CITRUS	4	80k
	ConvBig	Conv	DiffAI	7	1.8M
CIFAR10	ConvSmall	Conv	Standard	4	80k
	ConvSmall	Conv	DiffAI	4	80k
	ConvSmall	Conv	SABR	4	80k
	ConvSmall	Conv	CITRUS	4	80k
	ConvBig	Conv	DiffAI	7	2.5M
	ResNet-2B	ResNet	Standard	14	110K

D.2 Standard Accuracies for Evaluated DNNs:

Table 3: DNN Standard Accuracies

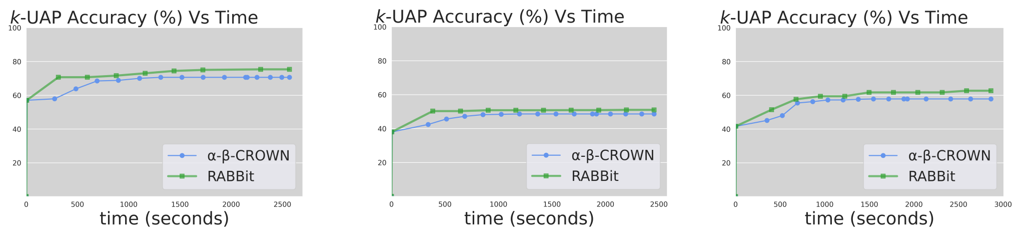
Dataset	Model	Train	Perturbation Bound (ϵ)	Accuracy (%)
CIFAR10	ConvSmall	Standard	1/255	62.9
	ConvSmall	DiffAI	5/255	45.9
	ConvSmall	SABR	2/255	63.6
	ConvSmall	CITRUS	2/255	63.9
	ConvBig	DiffAI	2/255	53.8
	ResNet-2B	Standard	1/255	67.5
MNIST	ConvSmall	Standard	0.10	32.5
	ConvSmall	DiffAI	0.13	32.5
	ConvSmall	SABR	0.15	48.7
	ConvSmall	CITRUS	0.15	48.6
	ConvBig	DiffAI	0.2	38.9

E Average Improvement in t^* with Strong Branching

Table 4: Average Improvement in t^* with Strong Bounding

Dataset	Network	Training	Perturbation	RACoon		α, β-CROWN	
	Structure	Method	Bound (ϵ)	Avg. Improvement (%)	95% CI	Avg. Improvement (%)	95% CI
CIFAR	ConvSmall	DiffAI	5/255	108.7	[93.9, 126.1]	102.5	[92.7, 115.4]
	ConvSmall	CITRUS	2/255	77.9	[75.3, 81.6]	86.9	[86.2, 88.1]
MNIST	ConvSmall	DiffAI	5/255	57.7	[55.5, 60.2]	54.4	[53.0, 56.0]
	ConvSmall	CITRUS	2/255	40.8	[39.8, 41.9]	37.1	[36.4, 37.8]

F MNIST k-UAP Verification Vs Time

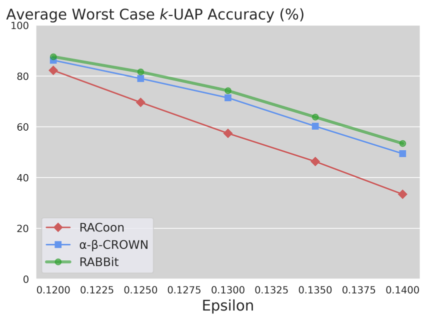


(a) DiffAI (MNIST) (b) SABR (MNIST) (c) CITRUS (MNIST)

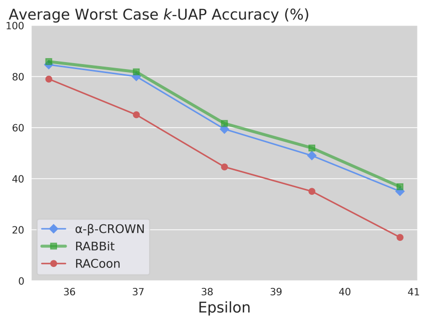
Figure 5: Average Worst-Case k-UAP Accuracy vs Time for ConvSmall MNIST DNNs.

G Additional k-UAP verification results for different ϵ, k, and k_t values

G.1 Different ϵ values for MNIST networks:



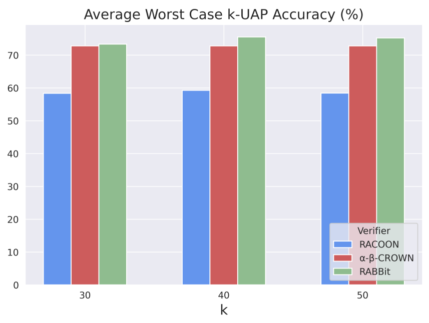
(a) DiffAI (MNIST)



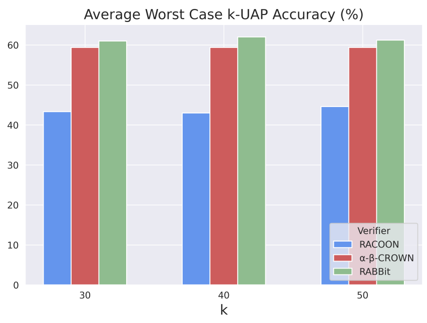
(b) CITRUS (MNIST)

Figure 6: Average worst case k-UAP accuracy vs ϵ for MNIST DNNs.

G.2 Different k values for MNIST networks:



(a) DiffAI (MNIST)



(b) CITRUS (MNIST)

Figure 7: Average Worst Case k-UAP accuracy for different k values for MNIST ConvSmall DNNs.

G.3 Different k_t values:

Table 5: Analysis of RABBit on MNIST for Different k_t values

Dataset	Network Structure	Training Method	Perturbation Bound (ϵ)	k_t 10	15	20
	ConvSmall	DiffAI	0.13	69.6	72.8	75.2
MNIST	ConvSmall	CITRUS	0.15	56.8	60.4	61.6

Table 6: Analysis of RABBit on CIFAR for Different k_t values

Dataset	Network Structure	Training Method	Perturbation Bound (ϵ)	k_t 8	9	10
	ConvSmall	DiffAI	5/255	58.8	59.6	59.8
CIFAR	ConvSmall	CITRUS	2/255	83.0	83.6	83.6

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: See Section 1 for main claims and contributions. The main claims made in this section and the abstract reflect the paper's scope and contributions.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: See end of Section 5 for the limitations.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification:

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: See experimental setup in Section 6.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide the code to replicate the main results of this paper.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please See Section the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See Section the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: See experimental setup in Section 6.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: RABBit is a deterministic verifier. The experiment "Evaluating Bound Improvement" (Section 6) is the only randomized experiment in the paper. We report the mean and 95% confidence intervals of the experiment in Appendix E.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: See experimental setup in Section 6.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Submission meets all ethical guidelines after authors reviewed the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: See Section 1 and Section 7 for societal impacts of the work.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: No data or models with high risk for misuse were used.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have only utilized publically available code, models, and datasets and properly cited all relevant works.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We provide the code to replicate the main results of this paper.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: No crowdsourcing nor human research with subject participants.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: No crowdsourcing nor human research with subject participants.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.