Scalable Relational Verification and Training for Deep Neural Networks

Debangshu Banerjee[1], Changming Xu[1], and Gagandeep Singh[1,2]

[1] University of Illinois Urbana-Champaign
[2] VMware Research

Abstract. We consider verifying relational properties defined over deep neural networks (DNNs) such as robustness against universal adversarial perturbations (UAP), certified worst-case hamming distance for binary string classifications, etc. Precise verification of these properties necessitates reasoning about multiple executions of the same DNN. However, most existing works in DNN verification only handle properties defined over single executions and as a result, are imprecise for relational properties. Though few recent works for relational DNN verification, capture linear dependencies between the inputs of multiple executions, they do not leverage dependencies between the outputs of hidden layers producing imprecise results. We develop a scalable relational verification framework that utilizes cross-execution dependencies at all layers of the DNN gaining substantial precision over SOTA baselines on a wide range of datasets, networks, and relational properties. Orthogonally, we propose a certifiable training method for relational properties to learn network parameters for facilitating relational verification on the trained network.

Keywords: Neural Network Verification · Relational Verification · Certifiable Training.

1 Introduction

Deep neural networks (DNNs) have gained widespread prominence across various domains, including safety-critical areas like autonomous driving [6] or medical diagnosis [1], etc. Especially in these domains, the decisions made by these DNNs hold significant importance, where errors can lead to severe consequences. However, due to the black-box nature and highly nonlinear behavior of DNNs, reasoning about them is challenging. Despite notable efforts in identifying and mitigating DNN vulnerabilities [18,31,36,44,59,52], these methods cannot guarantee safety. Consequently, significant research has been dedicated to formally verifying the safety properties of DNNs. Despite advancements, current DNN verification techniques can not handle relational properties prevalent in practical scenarios. Most of the existing efforts focus on verifying the absence of input-specific adversarial examples within the local neighborhood of test inputs. However, recent studies [27] highlight the impracticality of attacks targeting individual inputs. In practical attack scenarios [30,28,27], there is a trend towards

developing universal adversarial perturbations (UAPs) [36] designed to affect a significant portion of inputs from the training distribution. Since the same adversarial perturbation is applied to multiple inputs, the executions on different perturbed inputs are related, and exploiting the relationship between different executions is important for designing precise relational verifiers. Existing DNN verifiers working on individual executions lack these capabilities and as a result, lose precision. Beyond UAP verification, other relevant relational properties include measuring the worst-case hamming distance for binary string classification and bounding the worst-case absolute difference between the original number and the number classified using a digit classifier where inputs perturbed with common perturbation [45].

Key challenges: For precise relational verification, we need scalable algorithms to track the relationship between DNN's outputs across multiple executions. Although it is possible to exactly encode DNN executions with piecewise linear activation functions (e.g. ReLU) over input regions specified by linear inequalities as MILP (Mixed Integer Linear Program), the corresponding MILP optimization problem is computationally expensive. For example, MILP encoding of k executions of a DNN with n_r ReLU activations in the worst case introduces $O(n_r \times k)$ integer variables. Considering the cost of MILP optimization grows exponentially with the number of integer variables, even verifying small DNNs w.r.t a relational property defined over k execution with MILP is practically infeasible. For scalability, [23] completely ignores the dependencies across executions and reduces relational verification over k executions into k individual verification problems solving them independently. SOTA relational verifier [65] first obtains provably correct linear approximations of the DNN with existing non-relational verifier [61] without tracking any cross-execution dependencies then adds linear constraints at the input layer capturing linear dependencies between inputs used in different executions. In this case, ignoring cross-execution dependencies while computing provably correct linear approximations of the DNN for each execution leads to the loss of precision (as confirmed by our experiments in Section 6). This necessitates developing scalable algorithms for obtaining precise approximations of DNN outputs over multiple executions that benefit from cross-execution dependencies. Similarly, existing certified training methods work for training neural networks with properties defined over single executions. The training problem for the UAP robustness (Section 4.1) involves maximizing the expected loss under a single perturbation applied to multiple inputs. The maximization requires relational cross-executional reasoning. In this work, we lift these existing training methods for training neural networks with properties defined over k executions such as robustness against UAP.

Our contributions: We make the following contributions to improve the precision of relational DNN verification:

– In contrast to the SOTA baselines, we compute a provably correct parametric linear approximation of the DNN for each execution using parametric bounds of activation functions (e.g. ReLU) as done in existing works [62,46]. Instead of learning the parameters for each execution independently as done in [62],

we refine the parametric bounds corresponding to multiple executions together. In this case, the bound refinement at the hidden layer takes into account the cross-execution dependencies so that the learned bounds are tailored for verifying the specific *relational* property.
- For scalable cross-executional bound refinement, we (a) formulate a linear programming-based relaxation of the relational property, (b) find a provably correct differentiable closed form of the corresponding Dual function that preserves dependencies between parameters from different executions while being suitable for scalable differentiable optimization techniques, (c) using the differentiable closed form refine the parametric bound with scalable differential optimization methods (e.g. gradient descent).
- We develop RACoon (**R**elational DNN **A**nalyzer with **C**ross-Excuti**o**nal B**o**und Refi**n**ement) that formulates efficiently optimizable MILP instance with cross-executional bound refinement for precise relational verification.
- We perform extensive experiments on popular datasets, multiple DNNs (standard and robustly trained), and multiple relational properties showcasing that RACoon significantly outperforms the current SOTA baseline.
- Additionally we propose CITRUS (**C**ross-**I**nput certified **TR**aining for **U**niversal perturbation**S**) a certified training method for relational properties that exploits the dependencies between different executions while training. We show DNNs trained with CITRUS achieve SOTA clean and worst case UAP accuracy compared to existing non-relational certified training methods.

Outline: We discuss necessary background in Section 2, describe the key components of RACoon in Section 3 and CITRUS in Section 4. The experimental results and related works are in Section 6 and Section 7 respectively.

2 Background

We provide the necessary background on approaches for non-relational DNN verification, DNN safety properties that can be encoded as relational properties, and existing works on parametric bound refinement for individual executions. For certified training, we discuss existing training methods for input-specific adversarial perturbation.

Non-relational DNN verification: For individual execution, DNN verification involves proving that the network outputs $\mathbf{y} = N(\mathbf{x} + \boldsymbol{\delta})$ corresponding to all perturbations $\mathbf{x} + \boldsymbol{\delta}$ of an input $\mathbf{x}$ specified by ϕ, satisfy a logical specification ψ. For common safety properties like local DNN robustness, the output specification (ψ) is expressed 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, 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 $\mathbf{x} \in \phi_t \subseteq \mathbb{R}^{n_0}$ satisfying ϕ the following condition holds $\mathbf{L}^T\mathbf{x} + b \leq \mathbf{c}^T N(\mathbf{x})$. To show $\mathbf{c}^T N(\mathbf{x}) \geq 0$ for all $\mathbf{x} \in \phi_t$ DNN verifiers prove for all $\mathbf{x} \in \phi_t$, $\mathbf{L}^T\mathbf{x} + b \geq 0$ holds.

DNN relational properties: For a DNN $N : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_l}$, relational properties defined over k executions of N are specified by the tuple (Φ, Ψ) where the input specification $\Phi : \mathbb{R}^{n_0 \times k} \rightarrow \{true, false\}$ 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 $\Psi : \mathbb{R}^{n_l \times k} \rightarrow \{true, false\}$ 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 over multiple executions.

UAP verification: Given a DNN N, in a UAP attack, the adversary tries to find an adversarial perturbation with a bounded L_∞ norm that maximizes the misclassification rate of N when the same adversarial perturbation is applied to all inputs drawn from the input distribution. Conversely, the UAP verification problem finds the provably correct worst-case accuracy of N in the presence of a UAP adversary (referred to as UAP accuracy in the rest of the paper). [65] showed that it is possible to statistically estimate (Theorem 2 in [65]) UAP accuracy of N w.r.t input distribution provided we can characterize the UAP accuracy of N on k randomly selected images e.g. the k-UAP problem. For the rest of the paper, we focus on the k-UAP verification problem as improving the precision of k-UAP verification directly improves UAP accuracy on the input distribution. The k-UAP verification problem fundamentally differs from the commonly considered local L_∞ robustness verification where the adversary can perturb each input independently. Since the adversarial perturbation is common across a set of inputs, the UAP verification problem requires a relational verifier that can exploit the dependency between perturbed inputs. We provide the input specification Φ and the output specification Ψ of the UAP verification problem in Appendix A.1.

Worst case hamming distance: The hamming distance between two strings with the same length is the number of substitutions needed to turn one string into the other [21]. Given a DNN N, a binary string (a list of images of binary digits), we want to formally verify the worst-case bounds on the hamming distance between the original binary string and binary string recognized by N where a common perturbation can perturb each image of the binary digits. Common perturbations are a natural consequence of faulty input devices that uniformly distort the inputs already considered in verification problems in [40]. The input specification Φ and the output specification Ψ are in Appendix A.2. Beyond hamming distance and k-UAP, RACoon is a general framework capable

of formally analyzing the worst-case performance of algorithms that rely on multiple DNN executions [45]. For example, the absolute difference between the original and the number recognized by a digit classifier.

Parametric bound refinement: Common DNN verifiers [68,51] handle non-linear activations $\sigma(x)$ in DNN by computing linear lower bound $\sigma_l(x)$ and upper bound $\sigma_u(x)$ that contain all possible outputs of the activation w.r.t the input region ϕ_t i.e. for all possible input values x, $\sigma_l(x) \leq \sigma(x) \leq \sigma_u(x)$ holds. Common DNN verifiers including the SOTA relational verifier [65] also compute the linear bounds $\sigma_l(x)$ and $\sigma_u(x)$ statically without accounting for the property it is verifying. Recent works such as [62], instead of static linear bounds, use parametric linear bounds and refine the parameters with scalable differential optimization techniques to facilitate verification of the property (ϕ, ψ). For example, for $ReLU(x)$, the parametric lower bound is $ReLU(x) \geq \alpha \times x$ where the parameter $\alpha \in [0, 1]$ decides the slope of the lower bound. Since for any $\alpha \in [0, 1]$, $\alpha \times x$ is a valid lower bound of $ReLU(x)$ it is possible to optimize over α while ensuring mathematical correctness. Alternatively, [46] showed that optimizing α parameters is equivalent to optimizing the dual variables in the LP relaxed verification problem [58]. However, existing works can only optimize the α parameters w.r.t individual executions independently making these methods sub-optimal for relational verification. The key challenge here is to develop techniques for **jointly** optimizing α parameters over multiple DNN executions while leveraging their inter-dependencies.

Adversarial perturbations and robustness training: Given an input, output pair $(\mathbf{x}, y) \in \mathcal{X} \subseteq \mathbb{R}^{d_{\text{in}}} \times \mathbb{Z}$, and a classifier $f : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ which gives score, $f_k(\mathbf{x})$, for class k (let $\hat{f}(\mathbf{x}) = \arg\max_k f_k(x)$). An additive perturbation, $\mathbf{v} \in \mathbb{R}^{d_{\text{in}}}$, is adversarial for f on $\mathbf{x}$ if $\hat{f}(\mathbf{x} + \mathbf{v}) \neq y$. A classifier is adversarially robust on $\mathbf{x}$ for an l_p-norm ball, $\mathcal{B}_p(0, \epsilon)$, if it classifies all elements within the ball added to $\mathbf{x}$ to the correct class. Formally, $\forall \mathbf{v} \in B_p(0, \epsilon).\hat{f}(\mathbf{x} + \mathbf{v}) = y$. In this paper, we focus on l_∞-robustness, i.e. balls of the form $\mathcal{B}_\infty(\mathbf{x}, \epsilon) \coloneqq \{\mathbf{x}' = \mathbf{x} + \mathbf{v} | \|\mathbf{v}\|_\infty \leq \epsilon\}$, so will drop the subscript ∞. Note, our method can be extended to other l_p balls.

Training for robustness: For single-input robustness, we minimize the expected worst-case loss due to adversarial examples leading to the following optimization problem [37,33,32]:

$$\theta = \arg\min_{\theta} \mathop{\mathbb{E}}_{(\mathbf{x},y)\in\mathcal{X}} \left[\max_{\mathbf{x}' \in \mathcal{B}(\mathbf{x},\epsilon)} \mathcal{L}(f_\theta(\mathbf{x}'), y) \right] \tag{1}$$

Where $\mathcal{L}$ is a loss over the output of the DNN. As exactly solving the inner maximization is computationally impractical, in practice, it is approximated. Underapproximating the inner maximization is typically called adversarial training, a popular technique for obtaining good empirical robustness [32], but these techniques do not give good formal guarantees and are potentially vulnerable to stronger attacks [53]. We will focus on the second type of training, called certified training which overapproximates the inner maximization.

Certified training: One popular and effective method for non-relational verification is interval bound propagation (IBP) or Box propagation [34,19].

IBP first over-approximates the input region, $b^0 = \mathcal{B}(\mathbf{x}, \epsilon)$, as a Box $[\underline{\mathbf{x}}^0, \overline{\mathbf{x}}^0]$ where each dimension is an interval with center $\mathbf{x}$ and radius ϵ. Let our network, $f = L_1 \circ L_2 \circ \cdots \circ L_n$, be the composition of n linear or ReLU activation layers (for this paper we consider networks of this form, although our methods could be extended to other architectures). We then propagate the input region b^0 through each layer. For details of this propagation see [34,19]. At the output layer, we would like to show that the lower bound of the true class is greater than all upper bound of all other classes, i.e. $\forall i \in d_{\text{out}}, i \neq y. \overline{\mathbf{o}}_i - \underline{\mathbf{o}}_y < 0$. The IBP verification framework above adapts well to training. The Box bounds on the output can be encoded nicely into a loss function:

$$\mathcal{L}_{\text{IBP}}(\mathbf{x}, y, \epsilon) \coloneqq \ln\left(1 + \sum_{i \neq y} e^{\overline{\mathbf{o}}_i - \underline{\mathbf{o}}_y}\right) \tag{2}$$

To address the large approximation errors arising from Box analysis, SABR [37], a SOTA certified training method, obtains better standard and certified accuracy by propagating smaller boxes through the network. They do this by first computing an adversarial example, $\mathbf{x}' \in \mathcal{B}(\mathbf{x}, \epsilon - \tau)$ in a slightly truncated L_∞-norm ball. They then compute the IBP loss on a small ball around the adversarial example, $\mathcal{B}(\mathbf{x}', \tau)$, rather than on the entire ball, $\mathcal{B}(\mathbf{x}, \epsilon)$, where $\tau \ll \epsilon$.

$$\mathcal{L}_{\text{SABR}}(\mathbf{x}, y, \epsilon, \tau) \coloneqq \max_{x' \in \mathcal{B}(\mathbf{x}, \epsilon - \tau)} \mathcal{L}_{\text{IBP}}(\mathbf{x}', y, \tau) \tag{3}$$

Even though this is not a sound approximation of adversarial robustness, SABR accumulates fewer approximation errors due to its more precise Box analysis thus reduces overregularization improving standard/certified accuracy.

3 RACoon

3.1 Cross-executional bound refinement

Before delving into the details, first, we describe why it is essential to leverage cross-execution dependencies for relational verification. For illustrative purposes, we start with the k-UAP verification problem on a pair of executions i.e. $k = 2$. Note that bound refinement for worst-case hamming distance can be handled similarly. For 2-UAP, given a pair of unperturbed input $\mathbf{x_1}, \mathbf{x_2} \in \mathbb{R}^{n_0}$ first we want to prove whether there exists an adversarial perturbation $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ with bounded L_∞ norm $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$ such that N misclassifies both $(\mathbf{x_1} + \boldsymbol{\delta})$ and $(\mathbf{x_2} + \boldsymbol{\delta})$. Now, consider the scenario where both $\mathbf{x_1}$ and $\mathbf{x_2}$ have valid adversarial perturbations $\boldsymbol{\delta_1}$ and $\boldsymbol{\delta_2}$ but no *common* perturbation say $\boldsymbol{\delta}$ that works for both $\mathbf{x_1}$ and $\mathbf{x_2}$. In this case, 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. This highlights the necessity of utilizing cross-execution dependencies. Next, we detail three key steps for computing a provably correct parametric linear approximation of N

over multiple executions. So that the parameters from different executions are *jointly* optimized together to facilitate relational verification. Note that the SOTA relational verifier [65] statically computes linear approximations of N independently without leveraging any dependencies.

LP formulation: Let, N correctly classify $(\mathbf{x_1}+\boldsymbol{\delta})$ if $\mathbf{c_1}^T N(\mathbf{x_1}+\boldsymbol{\delta}) \geq 0$ and $(\mathbf{x_2}+\boldsymbol{\delta})$ if $\mathbf{c_2}^T N(\mathbf{x_2}+\boldsymbol{\delta}) \geq 0$ where $\mathbf{c_1}, \mathbf{c_2} \in \mathbb{R}^{n_l}$. Then N does not have a common adversarial perturbation iff for all $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$ the outputs $\mathbf{y_1} = N(\mathbf{x_1}+\boldsymbol{\delta})$ and $\mathbf{y_2} = N(\mathbf{x_2}+\boldsymbol{\delta})$ satisfy $\Psi(\mathbf{y_1}, \mathbf{y_2}) = (\mathbf{c_1}^T\mathbf{y_1} \geq 0) \vee (\mathbf{c_2}^T\mathbf{y_2} \geq 0)$. Any linear approximations specified with $\mathbf{L_1}, \mathbf{L_2} \in \mathbb{R}^{n_0}$ and $b_1, b_2 \in \mathbb{R}$ of N satisfying $\mathbf{L_1}^T(\mathbf{x_1}+\boldsymbol{\delta})+b_1 \leq \mathbf{c_1}^T\mathbf{y_1}$ and $\mathbf{L_2}^T(\mathbf{x_2}+\boldsymbol{\delta})+b_2 \leq \mathbf{c_2}^T\mathbf{y_2}$ for all $\boldsymbol{\delta}$ with $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$ allow us to verify the absence of common adversarial perturbation with the following LP (linear programming) formulation.

$$\begin{aligned} &min\ \ t \quad \text{s.t. } \|\boldsymbol{\delta}\|_\infty \leq \epsilon \\ &\mathbf{L_1}^T(\mathbf{x_1}+\boldsymbol{\delta})+b_1 \leq t,\ \mathbf{L_2}^T(\mathbf{x_2}+\boldsymbol{\delta})+b_2 \leq t \end{aligned} \tag{4}$$

Let t^* be the optimal solution of the LP formulation. Then $t^* \geq 0$ proves the absence of a common perturbation. For fixed linear approximations $\{(\mathbf{L_1}, b_1), (\mathbf{L_2}, b_2)\}$ of N, the LP formulation is exact i.e. it always proves the absence of common adversarial perturbation if it can be proved with $\{(\mathbf{L_1}, b_1), (\mathbf{L_2}, b_2)\}$ (see Theorem 1). This ensures that we do not lose any precision with the LP formulation and the LP formulation is more precise than any non-relational verifier using the same $\{(\mathbf{L_1}, b_1), (\mathbf{L_2}, b_2)\}$.

Theorem 1. *$\vee_{i=1}^{2}(\mathbf{L_i}^T(\mathbf{x_i}+\boldsymbol{\delta})+b_i \geq 0)$ holds for all $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ with $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$ if and only if $t^* \geq 0$.*

Proof: The proof follows from Appendix Theorem 6.

However, the LP formulation only works with fixed $\{(\mathbf{L_1}, b_1), (\mathbf{L_2}, b_2)\}$ and as a result, is not suitable for handling parametric linear approximations that can then be optimized to improve the relational verifier's precision. Instead, we use the equivalent Lagrangian Dual [7] which retains the benefits of the LP formulation while facilitating joint optimation of parameters from multiple executions as detailed below.

Dual with parametric linear approximations: Let, for a list of parametric activation bounds specified by a parameter list $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_m]$ we denote corresponding parametric linear approximation of N with the coefficient $\mathbf{L}(\boldsymbol{\alpha})$ and bias $\mathbf{b}(\boldsymbol{\alpha})$. First, for 2-UAP, we obtain $(\mathbf{L}_1(\boldsymbol{\alpha}_1), \mathbf{b}_1(\boldsymbol{\alpha}_1))$ and $(\mathbf{L}_2(\boldsymbol{\alpha}_2), \mathbf{b}_2(\boldsymbol{\alpha}_2))$ corresponding to the pair of executions using existing works [62]. For $i \in \{1, 2\}$, $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$ and $\mathbf{l}_i \preceq \boldsymbol{\alpha}_i \preceq \mathbf{u}_i$ the parametric linear bounds satisfy $\mathbf{L}_i(\boldsymbol{\alpha}_i)^T(\mathbf{x_i}+\boldsymbol{\delta}) + \mathbf{b}_i(\boldsymbol{\alpha}_i) \leq \mathbf{c_i}^T\mathbf{y_i}$ where $\mathbf{l}_i, \mathbf{u}_i$ are constant vectors defining valid range of the parameters $\boldsymbol{\alpha}_i$. For fixed $\boldsymbol{\alpha}_i$ the Lagrangian Dual of the LP formulation in Eq. 4 is as follows where $\lambda_1, \lambda_2 \in [0, 1]$ with $\lambda_1 + \lambda_2 = 1$ are the Lagrange multipliers relating linear approximations from different executions (details in Appendix D.1).

$$\max_{0 \leq \lambda_i \leq 1} \min_{\|\boldsymbol{\delta}\|_\infty \leq \epsilon} \sum\nolimits_{i=1}^{2} \lambda_i \times \left(\mathbf{L}_i(\boldsymbol{\alpha}_i)^T(\mathbf{x_i}+\boldsymbol{\delta}) + \mathbf{b}_i(\boldsymbol{\alpha}_i)\right)$$

Let, for fixed $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2$ the optimal solution of the dual formulation be $t^*(\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2)$. Then we can prove the absence of common perturbation provided the maximum value of $t^*(\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2)$ optimized over $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2$ is ≥ 0. This reduces the problem to the following: $\max t^*(\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2)$ s.t. $\mathbf{l}_1 \preceq \boldsymbol{\alpha}_1 \preceq \mathbf{u}_1 \quad \mathbf{l}_2 \preceq \boldsymbol{\alpha}_2 \preceq \mathbf{u}_2$. However, the optimization problem involves a max-min formulation and the number of parameters in $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2$ in the worst-case scales linearly with the number of activation nodes in N. This makes it hard to apply gradient descent-based techniques typically used for optimization [62]. Instead, we reduce the max-min formulation to a simpler maximization problem by finding an optimizable closed form of the inner minimization problem.

Deriving optimizable closed form : We want to characterize the closed form $G(\boldsymbol{\lambda}) = \min_{\|\boldsymbol{\delta}\|_\infty \leq \epsilon} \sum_{i=1}^{2} \lambda_i \times \left(\mathbf{L}_i(\boldsymbol{\alpha}_i)^T(\mathbf{x_i} + \boldsymbol{\delta}) + \mathbf{b}_i(\boldsymbol{\alpha}_i)\right)$ where $\boldsymbol{\lambda} = (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \lambda_1, \lambda_2)$ and use it for formulating the maximization problem. Note, $G(\boldsymbol{\lambda})$ is related to the dual function from optimization literature [7]. Naively, it is possible to solve the inner minimization problem for two different executions separately and then optimize them over $\overline{\boldsymbol{\alpha}} = (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2)$ using $\overline{G}(\overline{\boldsymbol{\alpha}}) = max(\overline{G}_1(\boldsymbol{\alpha}_1), \overline{G}_2(\boldsymbol{\alpha}_2))$ as shown below. However, $\overline{G}(\overline{\boldsymbol{\alpha}})$ produces a suboptimal result since it ignores cross-execution dependencies and misses out on the benefits of jointly optimizing $(\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2)$.

$$\overline{G}_i(\boldsymbol{\alpha}_i) = \min_{\|\boldsymbol{\delta}\|_\infty \leq \epsilon} \mathbf{L}_i(\boldsymbol{\alpha}_i)^T(\mathbf{x_i} + \boldsymbol{\delta}) + \mathbf{b}_i(\boldsymbol{\alpha}_i) \tag{5}$$

Since $\|\boldsymbol{\delta}\|_\infty$ is bounded by ϵ, it is possible to *exactly* compute the closed form of $G(\boldsymbol{\lambda})$ as shown below where for $j \in [n_0]$, $\mathbf{L}_i(\boldsymbol{\alpha}_i)[j] \in \mathbb{R}$ denotes the j-th component of $\mathbf{L}_i(\boldsymbol{\alpha}_i) \in \mathbb{R}^{n_0}$ and $a_i(\boldsymbol{\alpha}_i) = \mathbf{L}_i(\boldsymbol{\alpha}_i)^T\mathbf{x_i} + \mathbf{b}_i(\boldsymbol{\alpha}_i)$

$$G(\boldsymbol{\lambda}) = \min_{\|\boldsymbol{\delta}\|_\infty \leq \epsilon} \sum_{i=1}^{2} \lambda_i \times \left(\mathbf{L}_i(\boldsymbol{\alpha}_i)^T(\mathbf{x_i} + \boldsymbol{\delta}) + \mathbf{b}_i(\boldsymbol{\alpha}_i)\right)$$

$$G(\boldsymbol{\lambda}) = \sum_{i=1}^{2} \lambda_i \times a_i(\boldsymbol{\alpha}_i) + \min_{\|\boldsymbol{\delta}\|_\infty \leq \epsilon} \sum_{i=1}^{2} \lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i)^T\boldsymbol{\delta}$$

$$G(\boldsymbol{\lambda}) = \sum_{i=1}^{2} \lambda_i \times a_i(\boldsymbol{\alpha}_i) - \epsilon \times \sum_{j=1}^{n_0} \left| \sum_{i=1}^{2} \lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i)[j] \right|$$

Unlike $\overline{G}(\overline{\boldsymbol{\alpha}})$, $G(\boldsymbol{\lambda})$ relates linear approximations from two different executions using (λ_1, λ_2) enabling joint optimization over $(\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2)$. With the closed form $G(\boldsymbol{\lambda})$, we can use projected gradient descent to optimize $max_{\boldsymbol{\lambda}} G(\boldsymbol{\lambda})$ while ensuring the parameters in $\boldsymbol{\lambda}$ satisfy the corresponding constraints. Next, we provide theoretical guarantees about the correctness and efficacy of the proposed technique. For efficacy, we show the optimal solution $t^*(G)$ obtained with $G(\boldsymbol{\lambda})$ is always as good as $t^*(\overline{G})$ i.e. $t^*(G) \geq t^*(\overline{G})$ (Theorem 2) and characterize sufficient condition where $t^*(G)$ is strictly better i.e. $t^*(G) > t^*(\overline{G})$ (Appendix Theorem 9). Experiments substantiating the improvement in the optimal values ($t^*(G)$ vs. $t^*(\overline{G})$) are in Section 6.

Theorem 2. *If* $t^*(G) = \max_{\boldsymbol{\lambda}} G(\boldsymbol{\lambda})$ *and* $t^*(\overline{G}) = \max_{\boldsymbol{\alpha}_1,\boldsymbol{\alpha}_2} \overline{G}(\overline{\boldsymbol{\alpha}})$ *then* $t^*(\overline{G}) \leq t^*(G)$.

Proof: For any $\mathbf{l}_1 \preceq \boldsymbol{\alpha}_1 \preceq \mathbf{u}_1 \;\; \mathbf{l}_2 \preceq \boldsymbol{\alpha}_2 \preceq \mathbf{u}_2$, consider $\boldsymbol{\lambda}_1 = (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \lambda_1 = 1, \lambda_2 = 0)$ and $\boldsymbol{\lambda}_2 = (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \lambda_1 = 0, \lambda_2 = 1)$, then $G(\boldsymbol{\lambda}_1) = \min\limits_{\|\boldsymbol{\delta}\|_\infty \leq \epsilon} \mathbf{L}_1(\boldsymbol{\alpha}_1)^T(\mathbf{x_1} + \boldsymbol{\delta}) + \mathbf{b}_1(\boldsymbol{\alpha}_1)$ and $G(\boldsymbol{\lambda}_2) = \min\limits_{\|\boldsymbol{\delta}\|_\infty \leq \epsilon} \mathbf{L}_2(\boldsymbol{\alpha}_2)^T(\mathbf{x_2} + \boldsymbol{\delta}) + \mathbf{b}_2(\boldsymbol{\alpha}_2)$. Since, $t^*(G) \geq G(\boldsymbol{\lambda}_1)$ and $t^*(G) \geq G(\boldsymbol{\lambda}_2)$ then $t^*(G) \geq \max\limits_{1 \leq i \leq 2} G(\boldsymbol{\lambda}_i) = \overline{G}(\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2)$. Hence, $t^*(G) \geq \max_{\boldsymbol{\alpha}_1,\boldsymbol{\alpha}_2} \overline{G}(\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2) = t^*(\overline{G})$. The correctness proof for bound refinement between two executions is in Appendix D.1. Note that correctness does not necessitate the optimization technique to identify the global maximum, especially since gradient-descent-based optimizers may not always find the global maximum.

Genralization: Instead of a pair of executions considered above, we now generalize the approach to any set of n executions where $n \leq k$ (Appendix B). Until now, we assume for each execution the output specification is defined as a linear inequality i.e. $\mathbf{c_i}^T N(\mathbf{x_i} + \boldsymbol{\delta}) \geq 0$. Next, we generalize our method to any output specification for each execution defined with conjunction of m linear inequalities (Appendix C).

3.2 RACoon algorithm

The cross-executional bound refinement learns parameters over any set of n executions. However, for a relational property defined over k executions, since there are $2^k - 1$ non-empty subsets of executions, refining bounds for all possible subsets is impractical. Instead, we design a greedy heuristic to pick the subsets of executions so that we only use a small number of subsets for bound refinement.

Eliminating individually verified executions: First, we run existing non-relational verifiers [68,51] without tracking any dependencies across executions. RACoon eliminates the executions already verified with the non-relational verifier and does not consider them for subsequent steps. (lines 5 – 9 in Algo. 1) For example, for the k-UAP property, we do not need to consider those executions that are proved to have no adversarial perturbation $\boldsymbol{\delta}$ such that $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$. For relational properties considered in this paper, we formally prove the correctness of the elimination technique in Appendix Theorem 12 and showcase eliminating verified executions does not lead to any loss in precision of RACoon.

Greedy selection of unverified executions: For each execution that remains unverified with the non-relational verifier ($\mathcal{V}$), we look at $s_i = \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i}$ estimated by $\mathcal{V}$ where $\mathbf{y_i} = N(\mathbf{x_i} + \boldsymbol{\delta})$ and $\mathbf{c_{i,j}} \in \mathbb{R}^{n_l}$ defines the corresponding output specification $\psi^i(\mathbf{y_i}) = \bigwedge_{j=1}^{m} (\mathbf{c_{i,j}}^T \mathbf{y_i} \geq 0)$. Intuitively, for unverified executions, s_i measures the maximum violation of the output specification $\psi^i(\mathbf{y_i})$ and thus leads to the natural choice of picking executions with smaller violations for cross-executional refinement. We sort the executions in decreasing order of s_i and pick the first k_0 (hyperparameter) executions on input regions $\mathbf{X} = \{\phi_t^1, \ldots, \phi_t^{k_0}\}$ having smaller violations s_i where for all $i \in [k_0]$, $\phi_t^i = \{\mathbf{x'_i} + \boldsymbol{\delta} \mid \mathbf{x'_i},\ \boldsymbol{\delta} \in \mathbb{R}^{n_0} \wedge \|\boldsymbol{\delta}\|_\infty \leq \epsilon\}$ and $\mathbf{x'_i}$ is the unperturbed input. (line 11 of Algo. 1) In general, k_0 is a small constant i.e. $k_0 \leq 10$. Further, we limit

Algorithm 1 RACoon

1: **Input:** N, (Φ, Ψ), k, k_0, k_1, non-relational verifier $\mathcal{V}$.
2: **Output:** sound approximation of worst-case k-UAP accuracy or worst-case hamming distance $\mathbf{M}(\Phi, \Psi)$.
3: $I \leftarrow \{\}$. {Indices of executions not verified by $\mathcal{V}$}
4: $\mathcal{L} \leftarrow \{\}$ {Map storing linear approximations}
5: **for** $i \in [k]$ **do**
6: $(s_i, \mathbf{L}_i, b_i) \leftarrow \mathcal{V}(\phi^i, \psi^i)$.
7: **if** $\mathcal{V}$ can not verify (ϕ^i, ψ^i) **then**
8: $I \leftarrow I \cup \{i\}$; $\mathcal{L}[i] \leftarrow \mathcal{L}[i] \cup (\mathbf{L}_i, b_i)$.
9: $I_0 \leftarrow$ top-k_0 executions from I selected based on s_i.
10: **for** $\overline{I_0} \subseteq I_0$, $\overline{I_0} \neq \{\}$ and $|\overline{I_0}| \leq k_1$ **do**
11: $\mathcal{L}_{\overline{I_0}} \leftarrow$ CrossExcutionalRefinement$(\overline{I_0}, \Phi, \Psi)$.
12: $\mathcal{L} \leftarrow$ Populate$(\mathcal{L}, \mathcal{L}_{\overline{I_0}})$. {Storing $\mathcal{L}_{\overline{I_0}}$ in $\mathcal{L}$}
13: $\mathcal{M} \leftarrow$MILPFormulation$(\mathcal{L}, \Phi, \Psi, k, I)$.
14: **return** Optimize$(\mathcal{M})$.

the subset size to k_1 (hyperparameter) and do not consider any subset of $\mathbf{X}$ with a size more than k_1 for cross-executional bound refinement. (lines 12 – 15 in Algo. 1) Overall, we consider $\sum_{i=1}^{k_1} \binom{k_0}{i}$ subsets for bound refinement.

MILP formulation: RACoon MILP formulation involves two steps. First, we deduce linear constraints between the input and output of N for each unverified execution using linear approximations of N either obtained through cross-executional refinement or by applying the non-relational verifier. Secondly, similar to the current SOTA baseline [65] we encode the output specification Ψ as MILP objective that only introduces $O(k \times n_l)$ integer variables. Finally, we use an off-the-shelf MILP solver [20] to optimize the MILP.

For the i-th unverified execution, let $\phi_t^i = \{\mathbf{x'_i} + \boldsymbol{\delta} \mid \mathbf{x'_i},\ \boldsymbol{\delta} \in \mathbb{R}^{n_0} \wedge \|\boldsymbol{\delta}\|_\infty \leq \epsilon\}$ be the input region and for $\mathbf{y_i} = N(\mathbf{x'_i} + \boldsymbol{\delta})$, $\psi^i(\mathbf{y_i}) = \bigwedge_{i=1}^{m} (\mathbf{c_{i,j}}^T \mathbf{y_i} \geq 0)$ be the output specification. Subsequently for each clause $(\mathbf{c_{i,j}}^T \mathbf{y_i} \geq 0)$ in $\psi^i(\mathbf{y_i})$ let $\{(\mathbf{L}_{i,j}^1, b_{i,j}^1), \ldots, (\mathbf{L}_{i,j}^{k'}, b_{i,j}^{k'})\}$ be set of linear approximations. Then for each $l \in [k']$ we add the following linear constraints where $o_{i,j}$ is a real variable.

$$\mathbf{L}_{i,j}^l(\mathbf{x'_i} + \boldsymbol{\delta}) + b_{i,j}^l \leq o_{i,j}\ ;\ \|\boldsymbol{\delta}\|_\infty \leq \epsilon$$

Next, similar to [65] we encode output specification (ψ^i) as $z_i = (\min_{1 \leq j \leq m} o_{i,j}) \geq 0$ where $z_i \in \{0, 1\}$ are binary variables and $z_i = 1$ implies $\psi^i(y_i) = True$. Encoding of each ψ^i introduces $O(m)$ binary (integer) variables. Since for k-UAP and worst-case hamming distance, $m = n_l$ the total number of integer variables is in the worst case $O(k \times n_l)$. MILP encoding for k-UAP and worst-case hamming distance verification are shown in Appendix D.4. We prove the correctness of RACoon in Appendix Theorem 13 and show it is always at least as precise as [65] (Appendix Theorem 14). Worst-case time complexity analysis of RACoon is in Appendix E.

4 CITRUS

In this section, we introduce our formal training objective for certified UAP robustness. This objective is hard to compute so we instead seek to bound this objective. Lower bounds are akin to adversarial training and, as we show, in Section 6 do not obtain good certified accuracy against universal perturbations. Therefore, we define upper bounds that can be efficiently computed. These form the backbone for the CITRUS algorithm.

4.1 UAP Training Objective

Equation (1) gives the training objective for standard single-input adversarial robustness: minimizing the expected loss over the data distribution due to worst-case adversarial perturbations crafted separately for each input. For UAP robustness, we minimize the worst-case expected loss from a single perturbation applied to all points in the data distribution.

$$\theta = \arg\min_{\theta} \max_{\mathbf{u} \in \mathcal{B}(0,\epsilon)} \left(\mathop{\mathbb{E}}_{(x,y)\in\mathcal{X}} [\mathcal{L}(f_\theta(\mathbf{x}+\mathbf{u}), y)] \right) \tag{6}$$

Here, since UAPs are input-agnostic we maximize the expected value over $\mathbf{u} \in \mathcal{B}(\mathbf{0}, \epsilon)$. Solving this maximization exactly is computationally impractical [36,43]. To create an efficient training algorithm for certified UAP robustness we need an efficiently computable upper bound for the maximization. It is necessary to reduce the approximation error due to the upper bound as existing research has shown that training with a looser upper bound leads to more regularization and a reduction in clean accuracy [37].

4.2 k-Common Perturbations

In this section, we introduce the idea of k-common perturbations (k-cp) and show that the maximization of Equation (6) can be upper bounded by the expected loss over the dataset due to the worst-case k-cp (Theorem 3). By doing this, we get a set of losses based on k-cps. In Section 5 we use the upper bounds derived in this section to create an algorithm for certified UAP training which we show experimentally (Section 6.2) reduces regularization and increases accuracy while also achieving certified UAP accuracy.

We first define a boolean predicate, $A_f : \mathbb{R}^{d_{\text{in}}} \times \mathbb{N} \to \{true, false\}$ which is true when f is adversarial for $\mathbf{x}, y$:

$$A_f(\mathbf{x}, y) = \begin{cases} true & \text{if } \hat{f}(\mathbf{x}) \neq y \\ false & \text{otherwise} \end{cases}$$

Definition 1. $\mathbf{u} \in \mathbb{R}^{d_{in}}$ *is a* k*-common perturbation on a data distribution* $\mathcal{X}$ *for a network* f*, if there exists a set of* k *inputs for which* $\mathbf{u}$ *is adversarial. That is, if* $\exists\{(\mathbf{x}_i, y_i) | i \in [k]\} \subseteq \mathcal{X}. \bigwedge_{i=1}^{k} A_f(\mathbf{x}_i + \mathbf{u}, y_i)$.

Note that if $\mathbf{u}$ is a k-cp than $\mathbf{u}$ is also a $(k-1)$-cp, ..., 1-cp. Next, we define a boolean predicate function $\Psi_{\mathcal{X},f} : \mathbb{R}^{d_{\text{in}}} \times \mathbb{N} \to \{true, false\}$ which computes whether a given perturbation $\mathbf{u}$ is a k-cp w.r.t. $\mathcal{X}$ and f

$$\Psi_{\mathcal{X},f}(\mathbf{u}, k) = \begin{cases} true & \text{if } \exists\{(\mathbf{x}_i, y_i) | i \in [k]\} \subseteq \mathcal{X}. \bigwedge_{i=1}^{k} A_f(\mathbf{x}_i + \mathbf{u}, y_i) \\ false & \text{otherwise} \end{cases}$$

Since Ψ is monotonic w.r.t. k, there is a unique j for each $\mathbf{u}$ where $\Psi_{\mathcal{X},f}(\mathbf{u}, j)$ transitions from $1 \to 0$, i.e. $\exists j \in \mathbb{N}$ s.t. $\forall i \leq j. \Psi_{\mathcal{X},f}(\mathbf{u}, i)$ and $\forall i > j. \Psi_{\mathcal{X},f}(\mathbf{u}, i)$.

Definition 2. *$\hat{\Psi}_{\mathcal{X},f} : \mathbb{R}^{d_{in}} \to \mathbb{N}$ is a function which computes the transition point of $\Psi_{\mathcal{X},f}$ for $\mathbf{u}$. That is, $\hat{\Psi}_{\mathcal{X},f}(\mathbf{u}) = \arg\max_{k \in \mathbb{N}} k \cdot \Psi_{\mathcal{X},f}(\mathbf{u}, k)$.*

For ease of notation, we refer to $\Psi(\cdot) \coloneqq \Psi_{\mathbf{X},f}(\cdot)$ and $\hat{\Psi}(\cdot) \coloneqq \hat{\Psi}_{\mathbf{X},f}(\cdot)$.

Definition 3. *A k-cp set, $\mathcal{C}_{\mathcal{X},f}(k, \epsilon)$, is all perturbations in $\mathcal{B}(\mathbf{0}, \epsilon)$ that are k-cps w.r.t. $\mathcal{X}$ and f. That is, $\mathcal{C}_{\mathcal{X},f}(k, \epsilon) = \{\mathbf{u} | \mathbf{u} \in \mathcal{B}(\mathbf{0}, \epsilon), \Psi(\mathbf{u}, k)\}$.*

Let $\mathbf{u}^* \in \mathcal{B}(\mathbf{0}, \epsilon)$ be the point at which the expectation of Equation (6) is maximized (without loss of generality, we assume $\mathbf{u}^*$ is unique). Formally,

$$\mathop{\mathbb{E}}_{(x,y)\in\mathcal{X}} [\mathcal{L}(f(\mathbf{x} + \mathbf{u}^*), y)] = \max_{\mathbf{u}\in\mathcal{B}(0,\epsilon)} \left(\mathop{\mathbb{E}}_{(x,y)\in\mathcal{X}} [\mathcal{L}(f(\mathbf{x} + \mathbf{u}), y)] \right) \tag{7}$$

If $\mathbf{u}^*$ is a k-cp then the loss for an individual input when perturbed by $\mathbf{u}^*$ is bounded by maximizing the loss for that input over the entire k-cp perturbation set (Lemma 5). We also know that all j-cps are k-cps for $k < j$ so $\mathcal{C}(j, \epsilon) \subseteq \mathcal{C}(k, \epsilon)$ (Lemma 6). These two lemmas allow us to bound $\mathcal{L}(f(\mathbf{x} + \mathbf{u}^*), y)$ for each pair $(\mathbf{x}, y)$. Formal statements and proofs of these lemmas can be seen in Appendix G.1. We can now show that the maximization in Equation (6) can be upper bounded by an increasing sequence of values. Each value is an expectation, over the full data distribution, of the worst-case loss achievable by a k-cp. Formally,

Theorem 3. *Given $\mathcal{X} \subseteq \mathbb{R}^{d_{in}} \times \mathbb{N}$, network $f : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$, $\mathbf{u}^*$ as defined in Equation (7), and norm-bound $\epsilon \in \mathbb{R}$. Let $\kappa^* = \hat{\Psi}_{\mathcal{X},f}(\mathbf{u}^*)$ and $\mathcal{E}(k, \epsilon) = \mathbb{E}_{(x,y)\in\mathcal{X}} \left[\max_{\mathbf{u}\in\mathcal{C}_{\mathcal{X},f}(k,\epsilon)} \mathcal{L}(f(\mathbf{x} + \mathbf{u}), y)\right]$, then*

$$\max_{\mathbf{u}\in\mathcal{B}(0,\epsilon)} \left(\mathop{\mathbb{E}}_{(x,y)\in\mathcal{X}} [\mathcal{L}(f(\mathbf{x} + \mathbf{u}), y)] \right) \leq \mathcal{E}(\kappa^*, \epsilon) \leq \mathcal{E}(\kappa^* - 1, \epsilon) \leq \cdots \leq \mathcal{E}(1, \epsilon)$$

Proof Sketch. LHS equals $\mathbb{E}_{(x,y)\in\mathcal{X}} [\mathcal{L}(f(\mathbf{x} + \mathbf{u}^*), y)]$ via Equation (7) which is upperbounded by $\mathcal{E}(\kappa^*, \epsilon)$ by Lemma 5 applied to each element in the distribution. $\mathcal{E}(\kappa^*, \epsilon)$ is upperbounded by $\mathcal{E}(\kappa^* - 1, \epsilon), \ldots, \mathcal{E}(1, \epsilon)$ by Lemma 6 applied to each element in the distribution. Full proof is in Appendix G.1.

5 CITRUS

In this section, we will leverage Theorem 3 to develop our CITRUS algorithm for certified training against UAPs.

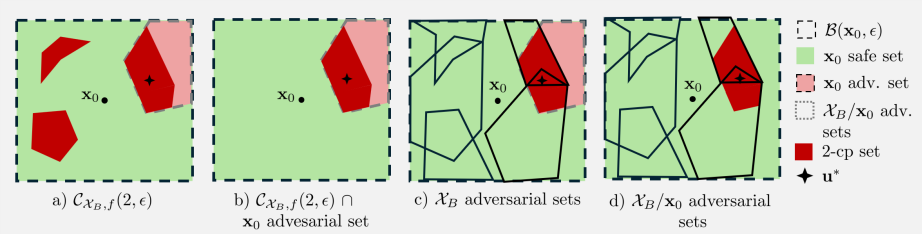


Fig. 1: Upper bounding $\mathcal{L}_{2\text{CP}}$. For each image, perturbations are shown on $\mathbf{x}_0$. a) shows the 2-cp set (■). b) shows the intersection of 2-cp set with the adversarial perturbation set for $\mathbf{x}_0$ (■). c) shows the adversarial perturbation set for all inputs in the batch (□, ■). d) shows the adversarial perturbation set for all inputs except for $\mathbf{x}_0$ (□)

5.1 *k*-CP Guided Losses

Theorem 3 gives us a sequence of upper bounds for the maximization of the UAP robustness objective. We will now focus on a batch-wise variant of these upper bounds for training. Given a batch of inputs $\mathcal{X}_B \subseteq \mathbb{R}^{d_{\text{in}}} \times \mathbb{Z}$, current input $(\mathbf{x}, y) \in \mathcal{X}_B$, and for each $k \in [\kappa^*]$ we get the following loss functions

$$\mathcal{L}_{k\text{CP}}(\mathcal{X}_B, \mathbf{x}, y, \epsilon) = \max_{\mathbf{u} \in \mathcal{C}_{\mathcal{X},f}(k,\epsilon)} \mathcal{L}(f(\mathbf{x}+\mathbf{u}), y) \tag{8}$$

However, in practice, computing κ^* is intractable as it would either require computing individual adversarial regions [13] and intersecting them or require finding $\mathbf{u}^*$ [36]. $\mathcal{L}_{k\text{CP}}$ is only an upper bound when $k \leq \kappa^*$ as if $k > \kappa^*$ then κ^* does not fall in the k-cp region.

Ideally, we would use the tightest upper bound ($\mathcal{L}_{\kappa^*\text{CP}}$), to induce the least amount of overapproximation; however, in practice, we do not know κ^*. For this paper, we consider $\mathcal{L}_{2\text{CP}}$ as it leads to efficient training algorithms. $\mathcal{L}_{1\text{CP}}$ is akin to standard certified training for single-input adversarial perturbations. Current research shows that networks trained by SOTA standard certified training can not eliminate single-input adversarial perturbations altogether [37,33,48], i.e. $\kappa^* \geq 1$ for these networks. Thus, when training networks to be certifiably robust to UAPs it is safe to assume that $\kappa^* \geq 1$. $\mathcal{L}_{2\text{CP}}$ penalizes perturbations affecting multiple inputs while still having a high chance of being an upper bound to the UAP robustness problem (if $\kappa^* = 1$ then we likely do not have any adversarial perturbations which affect multiple inputs and have succeded in training a network certifiably robust to UAPs).

5.2 CITRUS Loss

$\mathcal{L}_{2\text{CP}}$ is still not usable for training as it is expensive to compute $\mathcal{C}(2, \epsilon)$ (which would require intersecting single-input adversarial regions [13]). To make this usable, we first show that for a given input $\mathcal{L}_{2\text{CP}}$ can be upper bounded by

computing the maximum loss over the 1-cp perturbation sets from the other inputs in the batch. Formally, we have

Theorem 4. *Given $\mathcal{X}_B \subseteq \mathbb{R}^{d_{in}} \times \mathbb{Z}$, a network $f : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$, a given input $(\mathbf{x}_0, y_0)$, and norm-bound $\epsilon \in \mathbb{R}$. Then,*

$$\mathcal{L}_{2CP}(\mathcal{X}_B, \mathbf{x}_0, y_0, \epsilon) \leq \max_{\mathcal{C}_{\mathbf{u} \in \mathcal{X}/(\mathbf{x}_0, y_0), f}(1,\epsilon)} \mathcal{L}(f(\mathbf{x}_0 + \mathbf{u}), y_0)$$

Proof Sketch. Figure 1 gives a simplified visual guide for our proof, where $|\mathcal{X}_B| = 5$. The light red region (■) represents the set of adversarial perturbations for $\mathbf{x}_0$. In Figure 1 a) we see the 2-cp set (■) on top of $\mathbf{x}_0$, $\mathcal{L}_{2\text{CP}}$ maximizes the loss over ■. Assuming that we have a standard loss function where adversarial perturbations (■) have greater loss than safe perturbations (■) $(\forall \mathbf{v}, \mathbf{v}' \in \mathcal{B}(\mathbf{0}, \epsilon). \neg A_f(\mathbf{x}_0 + \mathbf{v}, y) \wedge A_f(\mathbf{x}_0 + \mathbf{v}', y) \implies \mathcal{L}(f(\mathbf{x}_0 + \mathbf{v}), y_0) \leq \mathcal{L}(f(\mathbf{x}_0 + \mathbf{v}'), y_0))$, then for $\mathbf{x}_0$ the maximum loss over ■ (LHS) occurs where ■ and ■ intersect (Figure 1 b). We can compute an upper bound on the LHS by maximizing over an overapproximation of the intersection set. This overapproximation can be computed by considering all the single-input adversarial perturbation sets (Figure 1 c). Finally, a key observation is that since we are trying to overapproximate the intersection set we do not have to include the set of adversarial perturbations for $\mathbf{x}_0$ reducing overregularization (Figure 1 d). This loss in overregularization is significant as seen in Appendix I. The formal proof can be found in Appendix G.2.

Although we cannot exactly compute $\mathcal{C}_{\mathcal{X}/(\mathbf{x},y),f}(1, \epsilon)$, SABR [37] shows that we can approximate this set in certified training with small boxes around a precomputed adversarial perturbation. This observation leads to our final loss.

Definition 4. *For batch $\mathcal{X}_B$, current input $(\mathbf{x}_i, y_i) \in \mathcal{X}_B$, norm-bound $\epsilon \in \mathbb{R}$, and small box norm-bound $\tau \in \mathbb{R}$, we define the CITRUS loss as*

$$\mathcal{L}_{CITRUS}(\mathcal{X}_B, \mathbf{x}_i, y_i, \epsilon, \tau) \coloneqq \sum_{(\mathbf{x}_j, y_j) \in \mathcal{X}_B, i \neq j} \mathcal{L}_{IBP}(\mathbf{x}_i + \arg\max_{\mathbf{v} \in \mathcal{B}(\mathbf{0}, \epsilon - \tau)} \mathcal{L}(\mathbf{x}_j, y_j), y_i, \tau)$$

5.3 CITRUS Algorithm

In this section, we show how to take the CITRUS loss function and turn it into an algorithm. Further details on box propagation can be found in Appendix H and a study on adding same-input boxes can be found in Appendix I.

From Loss to Algorithm. Figure 2 provides a simplified example to aid visual intuition for the CITRUS algorithm. On the LHS, we show the exact adversarial perturbation sets (■) for each input in the batch and view these sets collocated on the origin. As discussed in Section 5.2, it is expensive to compute the exact adversarial sets [13]. Instead, on the RHS, we approximate these regions with small bounding boxes (□) around cross-input adversarial perturbations (not including the same-input perturbation).

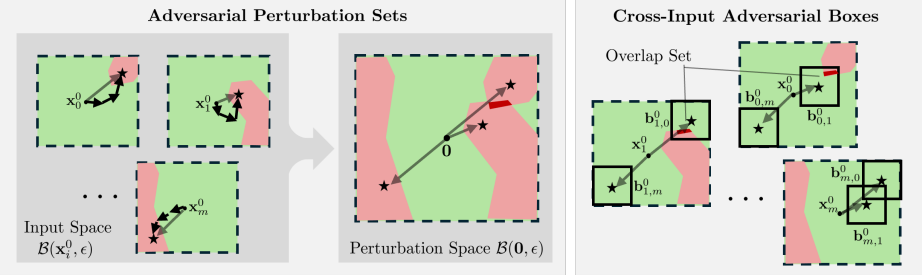


Fig. 2: Intuition behind CITRUS. UAPs occur where adversarial regions (■) from multiple inputs overlap (■). To approximate this, adversarial examples (⋆) are computed for each input, x_i, the corresponding perturbation vectors, v_i, (dark gray ➔) and adversarial regions are collocated to $\mathcal{B}(\mathbf{0}, \epsilon)$. To approximate ■ we take cross-input adversarial perturbations and draw l_∞ balls around them, $b^0_{i,j} = \mathcal{B}(\mathbf{x}_i + \mathbf{v}_j, \tau)$.

CITRUS Algorithm. In Algorithm 2, we show training with CITRUS. For each batch, $\mathcal{X}_B$, we first compute a set of adversarial perturbations $\mathbf{v}_i$ for each input $\mathbf{x}_i \in \mathcal{X}_B$ (Line 4). We then iterate through each input in the batch, $\mathbf{x}_i$ and through each cross-input adversarial perturbation, $\mathbf{v}_j$, not from the current input ($i \neq j$). For each input perturbation pair, we update the loss by computing the IBP loss on a box centered at $\mathbf{x}_i + \mathbf{v}_j$ with radius τ (Line 8). CITRUS is the first algorithm specialized for certified training against universal perturbations.

Algorithm 2 CITRUS Algorithm

1: Initialize θ
2: **for** each Epoch **do**
3: **for** $\mathcal{X}_B \subset \mathcal{X}$ **do**
4: Compute an adversarial perturbation $\mathbf{v}_i$ for $\mathbf{x}_i \in \mathcal{X}_B$
5: $\mathcal{L}_{\text{CITRUS}} \leftarrow 0$
6: **for** $i \in [|\mathcal{X}_B|]$ **do**
7: **for** $j \in [|\mathcal{X}_B|], j \neq i$ **do**
8: $\mathcal{L}_{\text{CITRUS}} = \mathcal{L}_{\text{CITRUS}} + \mathcal{L}_{\text{IBP}}(\mathbf{x}_i + \mathbf{v}_j, y_i, \tau)$
9: Update θ using $\mathcal{L}_{\text{CITRUS}}$

6 Experimental evaluation

6.1 RACoon evaluation

We evaluate the effectiveness of RACoon on a wide range of relational properties and a diverse set of DNNs and datasets. We consider the following relational properties: k-UAP, worst-case hamming distance as formally defined in Appendix A. The baselines we consider are the SOTA relational verifier [65] (referred to as I/O Formulation) and the non-relational verifier [61] from the SOTA auto_LiRPA

toolbox [61]. used by [65]. We also analyze the efficacy of cross-executional bound refinement in learning parametric bounds that can facilitate relational verification (Section 6.1). Note that we instantiate RACoon with the same non-relational verifier [61] used in I/O formulation [65].

Experiment setup and networks . We use standard convolutional architectures (ConvSmall, ConvBig, etc.) commonly seen in other neural network verification works [68,51] (see Table 1). Details of DNN architectures used in experiments are in Appendix F. We consider networks trained with standard training, robust training: DiffAI [35], CROWN-IBP [66], projected gradient descent (PGD) [31], and COLT [4]. We use pre-trained publically available DNNs: CROWN-IBP DNNs taken from the CROWN repository [66] and all other DNNs are from the ERAN repository [51]. The details regarding the frameworks RACoon uses, and the CPU and GPU information are in Appendix F.1.

Evaluating cross execution bound refinement Appendix Fig. 5 shows the values $t_i^*(G)$ and $t_i^*(\overline{G})$ after i-th iteration of Adam optimizer computed by cross-executional and individual refinement (using α-CROWN) respectively over a pair of executions (i.e. $k = 2$) on randomly chosen images. We used ConvSmall PGD and DiffAI DNNs trained on MNIST and CIFAR10 for this experiment. The ϵs used for MNIST PGD and DiffAI DNNs are 0.1 and 0.12 respectively while ϵs used for CIFAR10 PGD and DiffAI DNNs are 2.0/255 and 6.0/255 respectively. For each iteration i, $t_i^*(G) > t_i^*(\overline{G})$ shows that cross-executional refinement is more effective in learning parametric bounds that can facilitate relation verification. Since, for proving the absence of common adversarial perturbation, we need to show $t^* \geq 0$, in all 4 cases in Fig. 5 individual refinement fails to prove the absence of common adversarial perturbation while cross-executional refinement succeeds. Moreover, in all 4 cases, even the optimal solution of the LP (Eq. 14) formulated with linear approximations from individual refinement remains negative. For example, for MNIST DiffAI DNN, with LP, $t^*(\overline{G})$ improves to -0.05 from -0.2 but remains insufficient for proving the absence of common adversarial perturbation. This shows the importance of leveraging dependencies across executions.

Verification results: For k-UAP, both the baselines: non-relational verifier [61], I/O formulation [65] and RACoon computes a provably correct lower bound $\mathbf{M}(\Phi, \Psi)$ on the worst-case UAP accuracy. In this case, larger $\mathbf{M}(\Phi, \Psi)$ values produce a more precise lower bound tightly approximating the actual worst-case UAP accuracy. In contrast, for worst-case hamming distance $\mathbf{M}(\Phi, \Psi)$ is a provably correct upper bound and smaller $\mathbf{M}(\Phi, \Psi)$ values are more precise. Table 1 shows the verification results on different datasets (column 1), DNN architectures (column 3) trained with different training methods (column 4) where ϵ values defining L_∞ bound of $\boldsymbol{\delta}$ are in column 5. The relational properties: k-UAP and worst-case hamming distance on MNIST DNNs use $k = 20$ while k-UAP on CIFAR10 DNNs uses $k = 10$. For each DNN and ϵ, we run relational verification on k randomly selected inputs and repeat the experiment 10 times. We report worst-case UAP accuracy and worst-case hamming distance averaged

Table 1: RACoon Efficacy Analysis

Dataset	Property	Network Structure	Training Method	Perturbation Bound (ϵ)	Non-relational Verifier Avg. UAP Acc. (%)	Non-relational Verifier Avg. Time (sec.)	I/O Formulation Avg. UAP Acc. (%)	I/O Formulation Avg. Time (sec.)	RACoon Avg. UAP Acc. (%)	RACoon Avg. Time (sec.)
MNIST	UAP	ConvSmall	Standard	0.08	38.5	0.01	48.0	2.65	54.0 (+6.0)	5.20
	UAP	ConvSmall	PGD	0.10	70.5	0.21	72.0	0.92	77.0 (+5.0)	4.33
	UAP	IBPSmall	IBP	0.13	74.5	0.02	75.0	1.01	89.0 (+14.0)	2.01
	UAP	ConvSmall	DiffAI	0.13	56.0	0.01	61.0	1.10	68.0 (+7.0)	3.98
	UAP	ConvSmall	COLT	0.15	69.0	0.02	69.0	0.99	85.5 (+16.5)	2.68
	UAP	IBPMedium	IBP	0.20	80.5	0.1	82.0	0.99	93.5 (+11.5)	2.30
	UAP	ConvBig	DiffAI	0.20	81.5	1.85	81.5	2.23	91.5 (+10.0)	7.60
CIFAR10	UAP	ConvSmall	Standard	1.0/255	52.0	0.02	55.0	3.46	58.0 (+3.0)	7.22
	UAP	ConvSmall	PGD	3.0/255	21.0	0.01	26.0	1.57	29.0 (+3.0)	5.56
	UAP	IBPSmall	IBP	6.0/255	17.0	0.02	17.0	2.76	39.0 (+22.0)	6.76
	UAP	ConvSmall	DiffAI	8.0/255	16.0	0.01	20.0	2.49	30.0 (+10.0)	7.09
	UAP	ConvSmall	COLT	8.0/255	18.0	0.04	21.0	2.41	26.0 (+5.0)	11.02
	UAP	IBPMedium	IBP	3.0/255	46.0	0.15	50.0	2.13	71.0 (+21.0)	6.12
	UAP	ConvBig	DiffAI	3.0/255	17.0	1.33	20.0	3.42	25.0 (+5.0)	11.92
Dataset	Property	Network Structure	Training Method	Perturbation Bound (ϵ)	Non-relational Verifier Avg. Hamming distance	Non-relational Verifier Avg. Time (sec.)	I/O Formulation Avg. Hamming distance	I/O Formulation Avg. Time (sec.)	RACoon Avg. Hamming distance	RACoon Avg. Time (sec.)
MNIST	Hamming	ConvSmall	Standard	0.10	19.0	0.01	18.0	2.68	16.0 (-2.0)	4.43
	Hamming	ConvSmall	PGD	0.12	17.0	0.01	16.0	0.99	14.0 (-2.0)	3.20
	Hamming	ConvSmall	DiffAI	0.15	16.0	0.01	16.0	0.98	14.0 (-2.0)	3.46
	Hamming	IBPSmall	IBP	0.14	11.0	0.01	10.0	1.13	5.0 (-5.0)	2.56
	Hamming	ConvSmall	COLT	0.20	17.0	0.01	17.0	0.89	10.0 (-7.0)	1.88
	Hamming	IBPMedium	IBP	0.30	12.0	0.02	11.0	0.87	3.0 (-8.0)	1.75

over all 10 runs. Results in Table 1 substantiate that RACoon outperforms current SOTA baseline I/O formulation on all DNNs for both the relational properties. RACoon gains up to +16.5% and up to +22% improvement in the worst-case UAP accuracy (averaged over 10 runs) for MNIST and CIFAR10 DNNs respectively. Similarly, RACoon reduces the worst-case hamming distance (averaged over 10 runs) up to 8 which is up to 40% reduction.

Runtime analysis: Table 1 shows that RACoon is slower than I/O formulation. However, even for ConvBig architectures, RACoon takes less than 8 seconds (for 20 executions) for MNIST and takes less than 12 seconds (for 10 executions) for CIFAR10. The timings are much smaller compared to the timeouts allotted for similar architectures in the SOTA competition for verification of DNNs (VNN-Comp [8]) (200 seconds per execution).

6.2 CITRUS evaluation

We implemented CITRUS in Python [54] and PyTorch [39]. We compare CITRUS to existing SOTA single-input certified training methods. In Appendix L, we compare CITRUS to existing robust UAP training techniques. In Appendix M, we perform additional ablation studies on CITRUS.

Experimental Setup. We consider three popular image recognition datasets for certified training: MNIST [12], CIFAR-10 [25], and TinyImageNet [26]. We use a variety of challenging l_∞ perturbation bounds common in verification/robust training literature [63,56,51,50,48,37,33]. Unless otherwise indicated, we use a 7-layer convolutional architecture, CNN7, used in many prior works we compare against [48,37,33]. All experiments were performed on a desktop PC with a GeForce RTX(TM) 3090 GPU and a 16-core Intel(R) Core(TM) i9-9900KS CPU @ 4.00GHz. We use RACoon for worst-case UAP certification (note that this is an incomplete verifier so all results are underapproximations of true certified accuracy). Further evaluation, training/verification times, and training parameters can be found in Appendix K.

Table 2: Comparison of standard accuracy (Std) and certified average UAP accuracy (UCert) for different certified training methods on the full MNIST, CIFAR-10, and TinyImageNet test sets. A variation on [64] is used for certified average UAP accuracy.

Dataset	ϵ	Training Method	Source	Std [%]	UCert [%]
MNIST	0.1	IBP	Shi *et al.* [48]	98.84	98.12
		SABR	Müller *et al.* [37]	99.23	98.37
		TAPS	Mao *et al.* [33]	99.19	**98.65**
		CITRUS	this work	**99.27**	98.41
	0.3	IBP	Shi *et al.* [48]	97.67	94.76
		SABR	Müller *et al.* [37]	98.75	95.37
		TAPS	Mao *et al.* [33]	98.53	95.24
		CITRUS	this work	**99.04**	**95.61**
CIFAR-10	$\frac{2}{255}$	IBP	Shi *et al.* [48]	66.84	59.41
		SABR	Müller *et al.* [37]	79.24	65.38
		TAPS	Mao *et al.* [33]	79.76	66.62
		CITRUS	this work	**83.45**	**66.98**
	$\frac{8}{255}$	IBP	Shi *et al.* [48]	48.94	39.05
		SABR	Müller *et al.* [37]	52.38	**41.57**
		TAPS	Mao *et al.* [33]	52.82	40.90
		CITRUS	this work	**63.12**	39.88
TinyImageNet	$\frac{1}{255}$	IBP	Shi *et al.* [48]	25.92	18.50
		SABR	Müller *et al.* [37]	28.85	21.53
		TAPS	Mao *et al.* [33]	28.98	24.71
		CITRUS	this work	**35.62**	**26.27**

6.3 Main Results

We compare CITRUS to SOTA certified training methods in Table 2. Existing certified training methods train for single-input adversarial robustness. For each method, we report the best results achieved under any architecture presented in the respective paper. Across all datasets and ϵs we observe that CITRUS obtains better standard accuracy than existing methods (up to 10.3% increase for CIFAR-10 $\epsilon = 8/255$). CITRUS obtains SOTA performance for certified average UAP accuracy, obtaining better performance in 3 out of 5 cases than existing baselines (i.e. CITRUS obtains 26.27% certified average UAP accuracy on TinyImageNet vs. 24.71% from TAPS [33]) and comparable performance (CITRUS's certified average UAP accuracy is within 1.69% of SOTA baselines) on the rest. Our results show that CITRUS indeed decreases regularization and increases standard accuracy while maintaining good certified average UAP accuracy.

7 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 [17,50,51,49,68,61,62], (ii) complete verifiers that can always prove the property if it holds [55,17,9,10,3,14,15,16,57,38,2,67] and (iii) verifiers with probabilistic guarantees [11,29].

Relational DNN verifier: Existing DNN relational verifiers can be grouped into two main categories - (i) verifiers for properties (UAP, fairness, etc.) defined over multiple executions of the same DNN, [65,23], (ii) verifiers for properties (local DNN equivalence [41]) defined over multiple executions of different DNNs on the same input [41,42]. For relational properties defined over multiple executions of the same DNN the existing verifiers [23] reduce the verification problem into L_∞ robustness problem by constructing product DNN with multiple copies of the same DNN. However, the relational verifier in [23] treats all k executions of the DNN as independent and loses precision as a result of this. The SOTA DNN relational verifier [65] (referred to as I/O formulation in the rest of the paper) 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 as shown in our experiments. There exist, probabilistic verifiers, [60,69] based on randomized smoothing [11] for verifying relational properties. However, these works can only give probabilistic guarantees on smoothed models with high inference costs.

8 Conclusion

In this work, we present RACoon, a general framework for improving the precision of relational verification of DNNs through cross-executional bound refinement. Our experiments, spanning various relational properties, DNN architectures, and training methods demonstrate the effectiveness of utilizing dependencies across multiple executions. Furthermore, RACoon with cross-executional bound refinement proves to exceed the capabilities of the current state-of-the-art relational verifier [65]. Additionally, we propose CITRUS a certified training method for training DNNs with relational properties. We show that the CITRUS can leverage cross-executional dependencies while training producing higher clean and certified accuracies compared to existing SOTA certifiable robust training methods. While our focus has been on relational properties within the same DNN across multiple executions, both RACoon and CITRUS can be extended to properties involving different DNNs, such as local equivalence of DNN pairs [41] or properties defined over an ensemble of DNNs. We leave that as future work.

References

1. Amato, F., López, A., Peña-Méndez, E.M., Vaňhara, P., Hampl, A., Havel, J.: Artificial neural networks in medical diagnosis. Journal of Applied Biomedicine **11**(2) (2013)
2. Anderson, R., Huchette, J., Ma, W., Tjandraatmadja, C., Vielma, J.P.: Strong mixed-integer programming formulations for trained neural networks. Mathematical Programming (2020)
3. Bak, S., Tran, H., Hobbs, K., Johnson, T.T.: Improved geometric path enumeration for verifying relu neural networks. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12224, pp. 66–96. Springer (2020). `https://doi.org/10.1007/978-3-030-53288-8_4`, `https://doi.org/10.1007/978-3-030-53288-8_4`
4. Balunovic, M., Vechev, M.: Adversarial training and provable defenses: Bridging the gap. In: International Conference on Learning Representations (2020), `https://openreview.net/forum?id=SJxSDxrKDr`
5. Benz, P., Zhang, C., Karjauv, A., Kweon, I.S.: Universal adversarial training with class-wise perturbations. In: 2021 IEEE International Conference on Multimedia and Expo (ICME). pp. 1–6. IEEE (2021)
6. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)
7. Boyd, S., Vandenberghe, L.: Convex optimization. Cambridge university press (2004)
8. Brix, C., Müller, M.N., Bak, S., Johnson, T.T., Liu, C.: First three years of the international verification of neural networks competition (vnn-comp). International Journal on Software Tools for Technology Transfer pp. 1–11 (2023)
9. Bunel, R., Lu, J., Turkaslan, I., Kohli, P., Torr, P., Mudigonda, P.: Branch and bound for piecewise linear neural network verification. Journal of Machine Learning Research **21**(2020) (2020)
10. Bunel, R.R., Hinder, O., Bhojanapalli, S., Dvijotham, K.: An efficient nonconvex reformulation of stagewise convex optimization problems. Advances in Neural Information Processing Systems **33** (2020)
11. Cohen, J., Rosenfeld, E., Kolter, Z.: Certified adversarial robustness via randomized smoothing. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 1310–1320. PMLR (09–15 Jun 2019), `https://proceedings.mlr.press/v97/cohen19c.html`
12. Deng, L.: The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine **29**(6), 141–142 (2012)
13. Dimitrov, D.I., Singh, G., Gehr, T., Vechev, M.: Provably robust adversarial examples. In: International Conference on Learning Representations (2021)
14. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: International Symposium on Automated Technology for Verification and Analysis (2017)
15. Ferrari, C., Mueller, M.N., Jovanović, N., Vechev, M.: Complete verification via multi-neuron relaxation guided branch-and-bound. In: International Conference on Learning Representations (2022), `https://openreview.net/forum?id=l_amHf1oaK`

16. Fromherz, A., Leino, K., Fredrikson, M., Parno, B., Pasareanu, C.: Fast geometric projections for local robustness certification. In: International Conference on Learning Representations (2021), https://openreview.net/forum?id=zWy1uxjDdZJ
17. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: Ai2: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (SP) (2018)
18. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
19. Gowal, S., Dvijotham, K., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Arandjelovic, R., Mann, T., Kohli, P.: On the effectiveness of interval bound propagation for training verifiably robust models. arXiv e-prints pp. arXiv–1810 (2018)
20. Gurobi Optimization, LLC: Gurobi optimizer reference manual (2018)
21. Hamming, R.W.: Error detecting and error correcting codes. The Bell system technical journal **29**(2), 147–160 (1950)
22. Jovanovic, N., Balunovic, M., Baader, M., Vechev, M.: On the paradox of certified training. Transactions on Machine Learning Research (2022)
23. Khedr, H., Shoukry, Y.: Certifair: A framework for certified global fairness of neural networks. Proceedings of the AAAI Conference on Artificial Intelligence **37**(7), 8237–8245 (Jun 2023)
24. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
25. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
26. Le, Y., Yang, X.S.: Tiny imagenet visual recognition challenge (2015), https://api.semanticscholar.org/CorpusID:16664790
27. Li, J., Qu, S., Li, X., Szurley, J., Kolter, J.Z., Metze, F.: Adversarial music: Real world audio adversary against wake-word detection system. In: Proc. Neural Information Processing Systems (NeurIPS). pp. 11908–11918 (2019)
28. Li, J., Schmidt, F.R., Kolter, J.Z.: Adversarial camera stickers: A physical camera-based attack on deep learning systems. In: Proc. International Conference on Machine Learning, ICML. vol. 97, pp. 3896–3904 (2019)
29. Li, L., Zhang, J., Xie, T., Li, B.: Double sampling randomized smoothing. In: Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., Sabato, S. (eds.) Proceedings of the 39th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 162, pp. 13163–13208. PMLR (17–23 Jul 2022), https://proceedings.mlr.press/v162/li22aa.html
30. Liu, Z., Xu, C., Sie, E., Singh, G., Vasisht, D.: 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. pp. 1801–1817. USENIX Association (2023)
31. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: International Conference on Learning Representations (2018), https://openreview.net/forum?id=rJzIBfZAb
32. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: International Conference on Learning Representations (2018)
33. Mao, Y., Müller, M.N., Fischer, M., Vechev, M.: Taps: Connecting certified and adversarial training. arXiv e-prints pp. arXiv–2305 (2023)
34. Mirman, M., Gehr, T., Vechev, M.: Differentiable abstract interpretation for provably robust neural networks. In: Proc. International Conference on Machine Learning (ICML). pp. 3578–3586 (2018)

35. Mirman, M., Gehr, T., Vechev, M.: Differentiable abstract interpretation for provably robust neural networks. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 3578–3586. PMLR (10–15 Jul 2018), https://proceedings.mlr.press/v80/mirman18b.html
36. Moosavi-Dezfooli, S.M., Fawzi, A., Fawzi, O., Frossard, P.: Universal adversarial perturbations. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1765–1773 (2017)
37. Mueller, M.N., Eckert, F., Fischer, M., Vechev, M.: Certified training: Small boxes are all you need. In: The Eleventh International Conference on Learning Representations (2022)
38. Palma, A.D., Behl, H.S., Bunel, R.R., Torr, P.H.S., Kumar, M.P.: Scaling the convex barrier with active sets. In: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021 (2021)
39. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems **32** (2019)
40. Paterson, C., Wu, H., Grese, J., Calinescu, R., Păsăreanu, C.S., Barrett, C.: Deepcert: Verification of contextually relevant robustness for neural network image classifiers. In: Habli, I., Sujan, M., Bitsch, F. (eds.) Computer Safety, Reliability, and Security. pp. 3–17. Springer International Publishing, Cham (2021)
41. Paulsen, B., Wang, J., Wang, C.: Reludiff: Differential verification of deep neural networks. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. p. 714–726. ICSE '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3377811.3380337, https://doi.org/10.1145/3377811.3380337
42. Paulsen, B., Wang, J., Wang, J., Wang, C.: Neurodiff: Scalable differential verification of neural networks using fine-grained approximation. In: Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. p. 784–796. ASE '20, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3324884.3416560, https://doi.org/10.1145/3324884.3416560
43. Perolat, J., Malinowski, M., Piot, B., Pietquin, O.: Playing the game of universal adversarial perturbations. arXiv preprint arXiv:1809.07802 (2018)
44. Potdevin, Y., Nowotka, D., Ganesh, V.: An empirical investigation of randomized defenses against adversarial attacks. arXiv preprint arXiv:1909.05580 (2019)
45. Qin, C., Dvijotham, K.D., O'Donoghue, B., Bunel, R., Stanforth, R., Gowal, S., Uesato, J., Swirszcz, G., Kohli, P.: Verification of non-linear specifications for neural networks. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=HyeFAsRctQ
46. Salman, H., Yang, G., Zhang, H., Hsieh, C.J., Zhang, P.: A convex relaxation barrier to tight robustness verification of neural networks. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019), https://proceedings.neurips.cc/paper_files/paper/2019/file/246a3c5544feb054f3ea718f61adfa16-Paper.pdf
47. Shafahi, A., Najibi, M., Xu, Z., Dickerson, J., Davis, L.S., Goldstein, T.: Universal adversarial training. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 5636–5643 (2020)

48. Shi, Z., Wang, Y., Zhang, H., Yi, J., Hsieh, C.J.: Fast certified robust training with short warmup. Advances in Neural Information Processing Systems **34**, 18335–18349 (2021)
49. Singh, G., Ganvir, R., Püschel, M., Vechev, M.: Beyond the single neuron convex barrier for neural network certification. In: Advances in Neural Information Processing Systems (2019)
50. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.: Fast and effective robustness certification. Advances in Neural Information Processing Systems **31** (2018)
51. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proceedings of the ACM on Programming Languages **3**(POPL) (2019)
52. Sotoudeh, M., Thakur, A.V.: Abstract neural networks. In: Static Analysis: 27th International Symposium, SAS 2020, Virtual Event, November 18–20, 2020, Proceedings 27. pp. 65–88. Springer (2020)
53. Tramer, F., Carlini, N., Brendel, W., Madry, A.: On adaptive attacks to adversarial example defenses. Advances in neural information processing systems **33**, 1633–1645 (2020)
54. Van Rossum, G., Drake, F.L.: Python 3 Reference Manual. CreateSpace, Scotts Valley, CA (2009)
55. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: Advances in Neural Information Processing Systems (2018)
56. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.J., Kolter, J.Z.: Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. Advances in Neural Information Processing Systems **34** (2021)
57. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.J., Kolter, J.Z.: Beta-crown: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. arXiv preprint arXiv:2103.06624 (2021)
58. Wong, E., Kolter, J.Z.: Provable defenses against adversarial examples via the convex outer adversarial polytope. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, vol. 80, pp. 5283–5292. PMLR (2018), `http://proceedings.mlr.press/v80/wong18a.html`
59. Wu, H., Tagomori, T., Robey, A., Yang, F., Matni, N., Pappas, G., Hassani, H., Pasareanu, C., Barrett, C.: Toward certified robustness against real-world distribution shifts. In: 2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML). pp. 537–553. IEEE (2023)
60. Xie, C., Chen, M., Chen, P.Y., Li, B.: Crfl: Certifiably robust federated learning against backdoor attacks. In: International Conference on Machine Learning. pp. 11372–11382. PMLR (2021)
61. Xu, K., Shi, Z., Zhang, H., Wang, Y., Chang, K.W., Huang, M., Kailkhura, B., Lin, X., Hsieh, C.J.: Automatic perturbation analysis for scalable certified robustness and beyond. In: Proceedings of the 34th International Conference on Neural Information Processing Systems. NIPS'20, Curran Associates Inc., Red Hook, NY, USA (2020)
62. Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., Hsieh, C.J.: Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In: International Conference on Learning Representations (2021), `https://openreview.net/forum?id=nVZtXBI6LNn`

63. Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., Hsieh, C.J.: Fast and Complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In: International Conference on Learning Representations (2021), https://openreview.net/forum?id=nVZtXBI6LNn
64. Zeng, Y., Shi, Z., Jin, M., Kang, F., Lyu, L., Hsieh, C.J., Jia, R.: Towards robustness certification against universal perturbations. In: The Eleventh International Conference on Learning Representations (2022)
65. Zeng, Y., Shi, Z., Jin, M., Kang, F., Lyu, L., Hsieh, C.J., Jia, R.: Towards robustness certification against universal perturbations. In: The Eleventh International Conference on Learning Representations (2023), https://openreview.net/forum?id=7GEvPKxjtt
66. Zhang, H., Chen, H., Xiao, C., Gowal, S., Stanforth, R., Li, B., Boning, D., Hsieh, C.J.: Towards stable and efficient training of verifiably robust neural networks. In: Proc. International Conference on Learning Representations (ICLR) (2020)
67. Zhang, H., Wang, S., Xu, K., Li, L., Li, B., Jana, S., Hsieh, C.J., Kolter, J.Z.: General cutting planes for bound-propagation-based neural network verification. In: Oh, A.H., Agarwal, A., Belgrave, D., Cho, K. (eds.) Advances in Neural Information Processing Systems (2022), https://openreview.net/forum?id=5haAJAcofjc
68. Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient neural network robustness certification with general activation functions. Advances in neural information processing systems **31** (2018)
69. Zhang, Y., Albarghouthi, A., D'Antoni, L.: Bagflip: A certified defense against data poisoning. In: Oh, A.H., Agarwal, A., Belgrave, D., Cho, K. (eds.) Advances in Neural Information Processing Systems (2022), 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:

$$\Phi^\delta(\mathbf{x_1^*}, \ldots, \mathbf{x_k^*}) = \bigwedge_{(i,j \in [k]) \wedge (i<j)} (\mathbf{x_i^*} - \mathbf{x_j^*} = \mathbf{x_i} - \mathbf{x_j}) \tag{9}$$

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

$$\forall a \in [n_l].c_{i,j,a} = \begin{cases} 1 & \text{if } a \neq j \text{ and } a \text{ is the correct label for } \mathbf{y_i} \\ -1 & \text{if } a = j \text{ and } a \text{ is not the correct label for } \mathbf{y_i} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

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 2. 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

$$z_i(\boldsymbol{\delta}) = \begin{cases} 1 & \psi^i(N(\mathbf{x_i} + \boldsymbol{\delta})) \text{ is } True \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

$$\mu(\boldsymbol{\delta}) = \sum_{i=1}^{k} z_i(\boldsymbol{\delta}) \tag{12}$$

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

$$\mathbf{M}_0(\Phi, \Psi) = \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0},\ \|\boldsymbol{\delta}\| \leq \epsilon} \mu(\boldsymbol{\delta}) \tag{13}$$

A.2 Worst case Hamming distance verification

We consider a set of k unperturbed inputs $\mathbf{X} = \{\mathbf{x_1}, ..., \mathbf{x_k}\}$ where for all $i \in [k]$, $\mathbf{x_i} \in \mathbb{R}^{n_0}$, a peturbation budget $\epsilon \in \mathbb{R}$, and a binary digit classifier neural network $N_2 : \mathbb{R}^{n_0} \to \mathbb{R}^2$. We can define a binary digit string $\mathbf{S}^* \in \{0,1\}^k$ as a sequence of binary digits where each input $\mathbf{x_i}$ to N_2 is an image of a binary digit. We are interested in bounding the worst-case hamming distance between $\mathbf{S}$, the binary digit string classified by N_2, and $\mathbf{S}^*$ the actual binary digit string corresponding to the list of perturbed images $\forall i \in [k].\mathbf{x_i^*} = \mathbf{x_i} + \boldsymbol{\delta}$ s.t. $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ and $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$. Given these definitions, we can use the Φ, Ψ and $\mu(\boldsymbol{\delta})$ defined in section A.1 defined for k-UAP verification. In this case, the worst case hamming distance $\mathbf{M}_0(\Phi, \Psi)$ is defined as $\mathbf{M}_0(\Phi, \Psi) = k - \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0},\ \|\boldsymbol{\delta}\| \leq \epsilon} \mu(\boldsymbol{\delta})$.

B Genralization to multiple executions

Instead of a pair of executions considered above, we now generalize the approach to any set of n executions where $n \leq k$. With parametric linear approximations $\{(\mathbf{L}_1, b_1), \ldots, (\mathbf{L}_n, b_n)\}$ of N for all n executions, we formulate the following LP to prove the absence of common adversarial perturbation that works for *all* n executions. The proof of exactness of the LP formulation is in Appnedix Theorem 6.

$$\begin{aligned} &min\ \ t \quad \text{s.t. } \|\boldsymbol{\delta}\|_\infty \leq \epsilon \\ &\mathbf{L_i}^T(\mathbf{x_i} + \boldsymbol{\delta}) + b_i \leq t \qquad \forall i \in [n] \end{aligned} \tag{14}$$

Similar to a pair of executions, we first specify the Lagrangian dual of the LP (Eq. 14) by introducing n lagrangian multipliers $\lambda_1, \ldots, \lambda_n$ that satisfy for all $i \in [n]$ $\lambda_i \in [0,1]$ and $\sum_{i=1}^n \lambda_i = 1$. Subsequently, we obtain the closed form $G(\boldsymbol{\lambda})$ where $\boldsymbol{\lambda} = (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_n, \lambda_1, \ldots, \lambda_n)$ and $a_i(\boldsymbol{\alpha}_i) = \mathbf{L}_i(\boldsymbol{\alpha}_i)^T \mathbf{x_i} + \mathbf{b}_i(\boldsymbol{\alpha}_i)$ as shown below.

$$G(\boldsymbol{\lambda}) = \sum_{i=1}^{n} \lambda_i \times a_i(\boldsymbol{\alpha}_i) - \epsilon \times \sum_{j=1}^{n_0} \left| \sum_{i=1}^{n} \lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i)[j] \right|$$

Theoretical results regarding the correctness and efficacy of bound computation over n executions are in Appendix D.1.

C Genralization to a conjunction of linear inequalities

Until now, we assume for each execution the output specification is defined as a linear inequality i.e. $\mathbf{c_i}^T N(\mathbf{x_i} + \boldsymbol{\delta}) \geq 0$. Next, we generalize our method to any output specification for each execution defined with conjunction of m linear inequalities. For example, if $\mathbf{y_i}$ denotes the output of the i-th execution $\mathbf{y_i} = N(\mathbf{x_i} + \boldsymbol{\delta})$ then the output specification $\psi^i(\mathbf{y_i})$ is given by $\psi^i(\mathbf{y_i}) = \bigwedge_{j=1}^m (\mathbf{c_{i,j}}^T \mathbf{y_i} \geq$

0) where $\mathbf{c_{i,j}} \in \mathbb{R}^{n_l}$. In this case, $\psi(\mathbf{y_i})$ is satisfied iff $(\min_{1\leq j\leq m} \mathbf{c_{i,j}}^T \mathbf{y_i}) \geq 0$. Using this observation, we first reduce this problem to subproblems with a single linear inequality (see Appendix Theorem 10) and subsequently characterize the closed form $G(\boldsymbol{\lambda})$ for each subproblem separately. However, the number of subproblems in the worst case can be m^n which is practically intractable for large m and n. Hence, we greedily select which subproblems to use for bound refinement to avoid exponential blow-up in the runtime while ensuring the bound refinement remains provably correct (see Appendix D.2). Since most of the common DNN output specification can be expressed as a conjunction of linear inequalities [68] RACoon generalizes to them. Moreover, cross-excution bound refinement is not restricted to L_∞ input specification where $\|\boldsymbol{\delta}\|_\infty$ is bounded and can work for any $\|\cdot\|_p$ norm bounded perturbation (see Appendix D.3).

D Theorectical guarantees for cross-execution bound refinement

We obtain the theoretical guarantees of cross-execution bound refinement over n executions. Note that we do not show the theoretical guarantees for a pair of executions separately as it is just a special case with $n = 2$.

D.1 Theorectical guarantees for n of executions

Theorems for LP formulation First, we show the correctness of the LP formulation in Eq. 14 or for pair of execution in Eq. 4 (Theorem 5). We also show that for fixed linear approximations $\{(\mathbf{L_1}, b_1), \ldots, (\mathbf{L_n}, b_n)\}$ of N, the LP formulation is exact i.e. it always proves the absence of common adversarial perturbation if it does not exist (Theorem 6). In this case, $\Psi(\mathbf{y_1}, \ldots, \mathbf{y_n}) = \bigvee_{i=1}^{n} (\mathbf{c_i}^T \mathbf{y_i} \geq 0)$ where the outputs of N are $\mathbf{y_i} = N(\mathbf{x_i} + \boldsymbol{\delta})$. Let, t^* be the optimal solution of the LP in Eq. 14.

Lemma 1. $t^* = \min\limits_{\boldsymbol{\delta}\in\mathbb{R}^{n_0},\ \|\boldsymbol{\delta}\|_\infty \leq \epsilon} \max\limits_{1\leq i\leq n} \mathbf{L_i}^T(\mathbf{x_i} + \boldsymbol{\delta}) + b_i$.

Proof. $t^* = \min\limits_{\boldsymbol{\delta}\in\mathbb{R}^{n_0},\ \|\boldsymbol{\delta}\| \leq \epsilon} t(\boldsymbol{\delta})$ where if $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$ then $t(\delta)$ satisfies the following constraints $t(\delta) \geq \mathbf{L_i}^T(\mathbf{x_i} + \boldsymbol{\delta}) + b_i$ for all $i \in [n]$ then $t(\delta) \geq \max\limits_{1\leq i\leq n} \mathbf{L_i}^T(\mathbf{x_i} + \boldsymbol{\delta}) + b_i$. Let, $l^* = \min\limits_{\boldsymbol{\delta}\in\mathbb{R}^{n_0},\ \|\boldsymbol{\delta}\| \leq \epsilon} \max\limits_{1\leq i\leq n} \mathbf{L_i}^T(\mathbf{x_i} + \boldsymbol{\delta}) + b_i$.

$$t^* \geq \min_{\boldsymbol{\delta}\in\mathbb{R}^{n_0},\ \|\boldsymbol{\delta}\|_\infty \leq \epsilon} \max_{1\leq i\leq n} \mathbf{L_i}^T(\mathbf{x_i} + \boldsymbol{\delta}) + b_i = l^* \tag{15}$$

Next, we show that $l^* \geq t^*$. $l^* = \max_{1\leq i\leq n} \mathbf{L_i}^T(\mathbf{x_i} + \boldsymbol{\delta}^*) + b_i$ for some $\boldsymbol{\delta}^*$ where $\boldsymbol{\delta}^* \in \mathbb{R}^{n_0}$ and $\|\boldsymbol{\delta}^*\|_\infty \leq \epsilon$, then l^* satisfies the constraints $l^* \geq \mathbf{L_i}^T(\mathbf{x_i} + \boldsymbol{\delta}^*) + b_i$ for all $i \in [n]$. Since l^* is a valid feasible solution of the LP in Eq. 14 then $l^* \geq t^*$ as t^* is the optimal solution of the LP.

$l^* \geq t^*$ and from Eq. 15 $l^* \leq t^*$ implies $l^* = t^*$.

Theorem 5. *For all* $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ *and* $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$*, if for all* $i \in [n]$, $\mathbf{L_i}^T(\mathbf{x_i}+\boldsymbol{\delta})+b_i \leq \mathbf{c_i}^T\mathbf{y_i}$ *then* $(t^* \geq 0) \implies (\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.(\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies \Psi(\mathbf{y_1}, \ldots, \mathbf{y_n}))$ *holds.*

Proof. Since, for all $i \in [n]$, $\mathbf{L_i}^T(\mathbf{x_i}+\boldsymbol{\delta})+b_i \leq \mathbf{c_i}^T\mathbf{y_i}$, for all $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ and $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$, then $\min\limits_{\boldsymbol{\delta}\in\mathbb{R}^{n_0},\ \|\boldsymbol{\delta}\|_\infty\leq\epsilon} \max\limits_{1\leq i\leq n} \mathbf{L_i}^T(\mathbf{x_i}+\boldsymbol{\delta})+b_i \leq \min\limits_{\boldsymbol{\delta}\in\mathbb{R}^{n_0},\ \|\boldsymbol{\delta}\|_\infty\leq\epsilon} \max\limits_{1\leq i\leq n} \mathbf{c_i}^T\mathbf{y_i}$

$t^* = \min\limits_{\boldsymbol{\delta}\in\mathbb{R}^{n_0},\ \|\boldsymbol{\delta}\|_\infty\leq\epsilon} \max\limits_{1\leq i\leq n} \mathbf{L_i}^T(\mathbf{x_i}+\boldsymbol{\delta})+b_i \leq \min\limits_{\boldsymbol{\delta}\in\mathbb{R}^{n_0},\ \|\boldsymbol{\delta}\|_\infty\leq\epsilon} \max\limits_{1\leq i\leq n} \mathbf{c_i}^T\mathbf{y_i}$ Using lemma 1

$(t^* \geq 0) \implies \left(\min\limits_{\boldsymbol{\delta}\in\mathbb{R}^{n_0},\ \|\boldsymbol{\delta}\|_\infty\leq\epsilon} \max\limits_{1\leq i\leq n} \mathbf{c_i}^T\mathbf{y_i}\right) \geq 0$

$(t^* \geq 0) \implies (\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.(\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies \Psi(\mathbf{y_1}, \ldots, \mathbf{y_n}))$

Theorem 6. $\left(\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.(\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies \bigvee_{i=1}^n (\mathbf{L_i}^T(\mathbf{x_i}+\boldsymbol{\delta})+b_i \geq 0)\right)$ *holds if and only if* $t^* \geq 0$.

Proof. From lemma 1, $t^* = \min\limits_{\boldsymbol{\delta}\in\mathbb{R}^{n_0},\ \|\boldsymbol{\delta}\|_\infty\leq\epsilon} \max\limits_{1\leq i\leq n} \mathbf{L_i}^T(\mathbf{x_i}+\boldsymbol{\delta})+b_i$.

$$\begin{aligned}(t^* \geq 0) &\implies \left(\min_{\boldsymbol{\delta}\in\mathbb{R}^{n_0},\ \|\boldsymbol{\delta}\|_\infty\leq\epsilon} \max_{1\leq i\leq n} \mathbf{L_i}^T(\mathbf{x_i}+\boldsymbol{\delta})+b_i\right) \geq 0 \\ &\implies \left(\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.(\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies \bigvee_{i=1}^n (\mathbf{L_i}^T(\mathbf{x_i}+\boldsymbol{\delta})+b_i \geq 0)\right) \quad (16)\end{aligned}$$

$$\begin{aligned}(t^* < 0) &\implies \left(\min_{\boldsymbol{\delta}\in\mathbb{R}^{n_0},\ \|\boldsymbol{\delta}\|_\infty\leq\epsilon} \max_{1\leq i\leq 2} \mathbf{L_i}^T(\mathbf{x_i}+\boldsymbol{\delta})+b_i\right) < 0 \\ &\implies \left(\exists \boldsymbol{\delta} \in \mathbb{R}^{n_0}. \bigwedge_{i=1}^n (\mathbf{L_i}^T(\mathbf{x_i}+\boldsymbol{\delta})+b_i < 0) \bigwedge (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)\right) \\ \neg(t^* \geq 0) &\implies \neg\left(\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.(\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies \bigvee_{i=1}^n (\mathbf{L_i}^T(\mathbf{x_i}+\boldsymbol{\delta})+b_i \geq 0)\right) \quad (17)\end{aligned}$$

Using Eq. 16 and Eq. 17, $(t^* \geq 0) \iff \left(\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.(\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies \bigvee_{i=1}^n (\mathbf{L_i}^T(\mathbf{x_i}+\boldsymbol{\delta})+b_i \geq 0)\right)$.

Details for computing the Lagrangian Dual Next, we provide the details for computing the Lagrangian Dual of the LP formulation in Eq. 14. 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 $\min\limits_{t\in\mathbb{R},\|\boldsymbol{\delta}\|_\infty\leq\epsilon} (1 - \sum_{i=1}^n \lambda_i) \times t + \sum_{i=1}^n \lambda_i \times \left(\mathbf{L}_i^T(\mathbf{x_i}+\boldsymbol{\delta})+b_i\right) = -\infty$. This leads to

the following Lagrangian Dual form

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

For all $i\in[n]$, let parametric linear approximations of N are specified by $(\mathbf{L}_i(\boldsymbol{\alpha}_i),\mathbf{b}_i(\boldsymbol{\alpha}_i))$ then the Lagrangian Dual is as follows

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

Theorems for cross-execution bound refinement over n of executions Let, the $t^*_{appx}(G)$ denote the solution obtained by the optimization technique and $\boldsymbol{\lambda}^*_{appx}$ denote the value of $\boldsymbol{\lambda}$ corresponding to $t^*_{appx}(G)$. Note that $t^*_{appx}(G)$ can be different from global maximum $t^*(G)$ with $t^*(G)>t^*_{appx}(G)$. We show that if $t^*_{appx}(G)\geq 0$ then $\forall\boldsymbol{\delta}\in\mathbb{R}^{n_0}.(\|\boldsymbol{\delta}\|_\infty\leq\epsilon)\implies\Psi(\mathbf{y_1},\ldots\mathbf{y_n})$ holds where $\mathbf{y_i}=N(\mathbf{x_i}+\boldsymbol{\delta})$ for all $i\in[n]$. First, we prove the correctness of the characterization of $G(\boldsymbol{\lambda})$.

Lemma 2. *For all* $i\in[n]$, $0\leq\lambda_i\leq 1$, $\sum_{i=1}^{n}\lambda_i=1$, $\mathbf{l}_i\preceq\boldsymbol{\alpha}_i\preceq\mathbf{u}_i$, *if* $\boldsymbol{\lambda}=(\boldsymbol{\alpha}_1,\ldots,\boldsymbol{\alpha}_n,\lambda_1,\ldots,\lambda_n)$ *then* $\forall\boldsymbol{\delta}\in\mathbb{R}^{n_0}.(\|\boldsymbol{\delta}\|_\infty\leq\epsilon)\implies(G(\boldsymbol{\lambda})=\min_{\boldsymbol{\delta}\in\mathbb{R}^{n_0},\,\|\boldsymbol{\delta}\|_\infty\leq\epsilon}\sum_{i=1}^{n}\lambda_i\times\left(\mathbf{L}_i(\boldsymbol{\alpha}_i)^T(\mathbf{x_i}+\boldsymbol{\delta})+\mathbf{b}_i(\boldsymbol{\alpha}_i)\right)$ *where* $G(\boldsymbol{\lambda})=\sum_{i=1}^{n}\lambda_i\times a_i(\boldsymbol{\alpha}_i)-\epsilon\times\sum_{j=1}^{n_0}\left|\sum_{i=1}^{n}\lambda_i\times\mathbf{L}_i(\boldsymbol{\alpha}_i)[j]\right|$ *and* $a_i(\boldsymbol{\alpha}_i)=\mathbf{L}_i(\boldsymbol{\alpha}_i)^T\mathbf{x_i}+\mathbf{b}_i(\boldsymbol{\alpha}_i)$.

Proof. First we rewrite $G(\boldsymbol{\lambda})$ in Eq. 18 and find the closed form on $\min_{\boldsymbol{\delta}\in\mathbb{R}^{n_0},\,\|\boldsymbol{\delta}\|_\infty\leq\epsilon}\sum_{i=1}^{n}\lambda_i\times\mathbf{L}_i(\boldsymbol{\alpha}_i)^T\boldsymbol{\delta}$ in Eq. 21.

$$\sum_{i=1}^{n}\lambda_i\times\left(\mathbf{L}_i(\boldsymbol{\alpha}_i)^T(\mathbf{x_i}+\boldsymbol{\delta})+\mathbf{b}_i(\boldsymbol{\alpha}_i)\right)=\sum_{i=1}^{n}\lambda_i\times a_i(\boldsymbol{\alpha}_i)+\sum_{i=1}^{n}\lambda_i\times\mathbf{L}_i(\boldsymbol{\alpha}_i)^T\boldsymbol{\delta}$$

$$\min_{\boldsymbol{\delta}\in\mathbb{R}^{n_0},\,\|\boldsymbol{\delta}\|_\infty\leq\epsilon}\sum_{i=1}^{n}\lambda_i\times\left(\mathbf{L}_i(\boldsymbol{\alpha}_i)^T(\mathbf{x_i}+\boldsymbol{\delta})+\mathbf{b}_i(\boldsymbol{\alpha}_i)\right)=\sum_{i=1}^{n}\lambda_i\times a_i(\boldsymbol{\alpha}_i)+\min_{\boldsymbol{\delta}\in\mathbb{R}^{n_0},\,\|\boldsymbol{\delta}\|_\infty\leq\epsilon}\sum_{i=1}^{n}\lambda_i\times\mathbf{L}_i(\boldsymbol{\alpha}_i)^T\boldsymbol{\delta}\tag{18}$$

Now for fixed $\boldsymbol{\alpha}_i$, both $\mathbf{L}_i(\boldsymbol{\alpha}_i),\boldsymbol{\delta}\in\mathbb{R}^{n_0}$ are constant real vectors. Suppose for $j\in[n_0]$, $\mathbf{L}_i(\boldsymbol{\alpha}_i)[j]$ and $\boldsymbol{\delta}[j]$ denotes the j-th component of $\mathbf{L}_i(\boldsymbol{\alpha}_i)$ and $\boldsymbol{\delta}$ respectively.

Then,

$$\mathbf{L}_i(\boldsymbol{\alpha}_i)^T\boldsymbol{\delta} = \sum_{j=1}^{n_0} \mathbf{L}_i(\boldsymbol{\alpha}_i)[j] \times \boldsymbol{\delta}[j]$$

$$\sum_{i=1}^{n} \lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i)^T\boldsymbol{\delta} = \sum_{j=1}^{n_0} \left(\sum_{i=1}^{n} \lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i)[j] \right) \times \boldsymbol{\delta}[j]$$

$$-\epsilon \times \left| \sum_{i=1}^{n} \lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i)[j] \right| = \min_{-\epsilon \leq \boldsymbol{\delta}[j] \leq \epsilon} \left(\sum_{i=1}^{n} \lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i)[j] \right) \times \boldsymbol{\delta}[j] \tag{19}$$

$$\min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0},\, \|\boldsymbol{\delta}\|_\infty \leq \epsilon} \sum_{i=1}^{n} \lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i)^T\boldsymbol{\delta} = \sum_{j=1}^{n_0} \min_{-\epsilon \leq \boldsymbol{\delta}[j] \leq \epsilon} \left(\sum_{i=1}^{n} \lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i)[j] \right) \times \boldsymbol{\delta}[j] \tag{20}$$

$$\min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0},\, \|\boldsymbol{\delta}\|_\infty \leq \epsilon} \sum_{i=1}^{n} \lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i)^T\boldsymbol{\delta} = -\epsilon \times \sum_{j=1}^{n_0} \left| \sum_{i=1}^{n} \lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i)[j] \right| \text{ using Eq 19 and Eq. 20} \tag{21}$$

Combing Eq. 18 and Eq. 21

$$\min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0},\, \|\boldsymbol{\delta}\|_\infty \leq \epsilon} \sum_{i=1}^{n} \lambda_i \times \left(\mathbf{L}_i(\boldsymbol{\alpha}_i)^T(\mathbf{x_i} + \boldsymbol{\delta}) + \mathbf{b}_i(\boldsymbol{\alpha}_i) \right) = \sum_{i=1}^{n} \lambda_i \times a_i(\boldsymbol{\alpha}_i) - \epsilon \times \sum_{j=1}^{n_0} \left| \sum_{i=1}^{n} \lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i)[j] \right| = G(\boldsymbol{\lambda})$$

Theorem 7 (Correctness of bound refinement over n executions). *If* $t^*_{appx}(G) \geq 0$ *then* $(\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.(\|\boldsymbol{\delta}\| \leq \epsilon) \implies \Psi(\mathbf{y_1}, \ldots \mathbf{y_n}))$ *holds where* $\mathbf{y_i} = N(\mathbf{x_i} + \boldsymbol{\delta})$ *for all* $i \in [n]$.

Proof. $t^*_{appx}(G) = G(\boldsymbol{\lambda}^*_{appx})$ where $\boldsymbol{\lambda}^*_{appx} = (\boldsymbol{\alpha}^*_1, \ldots, \boldsymbol{\alpha}^*_n, \lambda^*_1, \ldots, \lambda^*_n)$ and for all $i \in [n]$, $\mathbf{l_i} \preceq \boldsymbol{\alpha}^*_i \preceq \mathbf{u_i}$, $0 \leq \lambda^*_i \leq 1$, $\sum_{i=1}^{n} \lambda^*_i = 1$. Then using lemma 2 we get

$$G(\boldsymbol{\lambda}^*_{appx}) = \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0},\, \|\boldsymbol{\delta}\|_\infty \leq \epsilon} \sum_{i=1}^{n} \lambda^*_i \times \left(\mathbf{L}_i(\boldsymbol{\alpha^*}_i)^T(\mathbf{x_i} + \boldsymbol{\delta}) + \mathbf{b}_i(\boldsymbol{\alpha^*}_i) \right) \tag{22}$$

Next we show that $G(\boldsymbol{\lambda}^*_{appx}) \leq \min_{\boldsymbol{\delta}\in\mathbb{R}^{n_0}, \|\boldsymbol{\delta}\|_\infty \leq \epsilon} \max_{1\leq i\leq n} \mathbf{c_i}^T\mathbf{y_i}$ where $\mathbf{y_i} = N(\mathbf{x_i}+\boldsymbol{\delta})$.

$$\big(\mathbf{L}_i(\boldsymbol{\alpha^\star}_i)^T(\mathbf{x_i}+\boldsymbol{\delta}) + \mathbf{b}_i(\boldsymbol{\alpha^\star}_i)\big) \leq \mathbf{c_i}^T\mathbf{y_i} \quad \forall i \in [n],\ \mathbf{y_i} = N(\mathbf{x_i}+\boldsymbol{\delta}) \text{ and } \|\boldsymbol{\delta}\|_\infty \leq \epsilon$$

$$\big(\mathbf{L}_i(\boldsymbol{\alpha^\star}_i)^T(\mathbf{x_i}+\boldsymbol{\delta}) + \mathbf{b}_i(\boldsymbol{\alpha^\star}_i)\big) \leq \max_{1\leq i\leq n} \mathbf{c_i}^T\mathbf{y_i} \qquad \forall i \in [n] \text{ and } \|\boldsymbol{\delta}\|_\infty \leq \epsilon$$

$$\sum_{i=1}^{n} \lambda_i^* \times \big(\mathbf{L}_i(\boldsymbol{\alpha^\star}_i)^T(\mathbf{x_i}+\boldsymbol{\delta}) + \mathbf{b}_i(\boldsymbol{\alpha^\star}_i)\big) \leq \max_{1\leq i\leq n} \mathbf{c_i}^T\mathbf{y_i} \times \sum_{i=1}^{n} \lambda_i^* \qquad \text{since } \forall i \in [n],\ \lambda_i^* \geq 0 \text{ and } \|\boldsymbol{\delta}\|_\infty \leq \epsilon$$

$$\sum_{i=1}^{n} \lambda_i^* \times \big(\mathbf{L}_i(\boldsymbol{\alpha^\star}_i)^T(\mathbf{x_i}+\boldsymbol{\delta}) + \mathbf{b}_i(\boldsymbol{\alpha^\star}_i)\big) \leq \max_{1\leq i\leq n} \mathbf{c_i}^T\mathbf{y_i} \qquad \text{since } \sum_{i=1}^{n} \lambda_i^* = 1 \text{ and } \|\boldsymbol{\delta}\|_\infty \leq \epsilon$$

$$G(\boldsymbol{\lambda}^*_{appx}) = \min_{\boldsymbol{\delta}\in\mathbb{R}^{n_0},\ \|\boldsymbol{\delta}\|_\infty \leq \epsilon} \sum_{i=1}^{n} \lambda_i^* \times \big(\mathbf{L}_i(\boldsymbol{\alpha^\star}_i)^T(\mathbf{x_i}+\boldsymbol{\delta}) + \mathbf{b}_i(\boldsymbol{\alpha^\star}_i)\big) \leq \min_{\boldsymbol{\delta}\in\mathbb{R}^{n_0}, \|\boldsymbol{\delta}\|_\infty \leq \epsilon} \max_{1\leq i\leq n} \mathbf{c_i}^T\mathbf{y_i} \tag{23}$$

Using Eq. 23 we show that

$$(t^*_{appx}(G) \geq 0) \implies (G(\boldsymbol{\lambda}^*_{appx}) \geq 0) \implies \left(\min_{\boldsymbol{\delta}\in\mathbb{R}^{n_0}, \|\boldsymbol{\delta}\|_\infty \leq \epsilon} \max_{1\leq i\leq n} \mathbf{c_i}^T\mathbf{y_i}\right) \geq 0$$
$$\implies (\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.(\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies \Psi(\mathbf{y_1}, \ldots \mathbf{y_n}))$$

Similar to Theorem 2, we show the optimal solution $t^*(G)$ obtained with $G(\boldsymbol{\lambda})$ is always as good as $t^*(\overline{G})$ i.e. $t^*(G) \geq t^*(\overline{G})$ for n executions.

Theorem 8. *If* $t^*(G) = \max_{\boldsymbol{\lambda}} G(\boldsymbol{\lambda})$ *and* $t^*(\overline{G}) = \max_{\boldsymbol{\alpha}_1,\ldots,\boldsymbol{\alpha}_n} \overline{G}(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_n)$ *then* $t^*(\overline{G}) \leq t^*(G)$.

Proof. For any $(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_n)$ satisfying $\mathbf{l}_i \preceq \boldsymbol{\alpha}_i \preceq \mathbf{u}_i$ for all $i \in [n]$, we consider $\boldsymbol{\lambda}_i = (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_n, \lambda_1 = 0, \ldots, \lambda_i = 1, \ldots, \lambda_n = 0)$. Then $G(\boldsymbol{\lambda}_i) = \min_{\|\boldsymbol{\delta}\|_\infty \leq \epsilon} \mathbf{L}_i(\boldsymbol{\alpha}_i)^T(\mathbf{x_i} + \boldsymbol{\delta}) + \mathbf{b}_i(\boldsymbol{\alpha}_i)$. Since, $t^*(G) \geq G(\boldsymbol{\lambda}_i)$ for all $i \in [n]$ then $t^*(G) \geq \max_{1\leq i\leq n} G(\boldsymbol{\lambda}_i) = \overline{G}(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_n)$. Hence, $t^*(G) \geq \max_{\boldsymbol{\alpha}_1,\ldots\boldsymbol{\alpha}_n} \overline{G}(\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2) = t^*(\overline{G})$.

Next, we characterize one sufficient condition where $t^*(G)$ is strictly better i.e. $t^*(G) > t^*(\overline{G})$. Note that Theorem 9 shows one possible case where $t^*(G)$ is strictly better and not the only possible condition where $t^*(G) > t^*(\overline{G})$ i.e. it is not necessary hold if $t^*(G) > t^*(\overline{G})$. Let, $(\boldsymbol{\alpha}^*_1, \ldots, \boldsymbol{\alpha}^*_n)$ be the optimal parameters corresponding to $t^*(\overline{G})$.

Theorem 9. *If for all* $i \in [n]$ *there exists* $j \in [n]$ *such that* $(a_j(\boldsymbol{\alpha^\star}_j) - a_i(\boldsymbol{\alpha^\star}_i)) > \epsilon \times (\|\mathbf{L}_j(\boldsymbol{\alpha^\star}_j)\|_1 - \|\mathbf{L}_i(\boldsymbol{\alpha^\star}_i)\|_1)$ *or* $2 \times \|\mathbf{L}_i(\boldsymbol{\alpha^\star}_i)\|_1 - \|\mathbf{L}_i(\boldsymbol{\alpha^\star}_i) + \mathbf{L}_j(\boldsymbol{\alpha^\star}_j)\|_1 > \frac{a_i(\boldsymbol{\alpha^\star}_i)}{\epsilon} - \frac{a_j(\boldsymbol{\alpha^\star}_j)}{\epsilon}$ *holds then* $t^*(G) > t^*(\overline{G})$.

Proof. Since $t^*(\overline{G}) = \max_{1\leq i\leq k} \min_{\boldsymbol{\delta}\in\mathbb{R}^{n_0}, \|\boldsymbol{\delta}\|_\infty \leq \epsilon} \mathbf{L}_i(\boldsymbol{\alpha^\star}_i)^T\boldsymbol{\delta} + a_i(\boldsymbol{\alpha^\star}_i) = \max_{1\leq i\leq k} -\epsilon \times \left(\sum_{j=1}^{n_0} |\mathbf{L}_i(\boldsymbol{\alpha^\star}_i)[j]|\right) + a_i(\boldsymbol{\alpha^\star}_i)$. This implies $t^*(\overline{G}) = \max_{1\leq i\leq k} -\epsilon \times \|\mathbf{L}_i(\boldsymbol{\alpha^\star}_i)\|_1 + a_i(\boldsymbol{\alpha^\star}_i)$. Now for any $i_0 \in [n]$

if $t^*(\overline{G}) = -\epsilon \times \|\mathbf{L}_{i_0}(\boldsymbol{\alpha^*}_{i_0})\|_1 + a_{i_0}(\boldsymbol{\alpha^*}_{i_0})$ (there exists at least one such i_0) then

$$
\begin{aligned}
&-\epsilon \times \|\mathbf{L}_{i_0}(\boldsymbol{\alpha^*}_{i_0})\|_1 + a_{i_0}(\boldsymbol{\alpha^*}_{i_0}) \geq -\epsilon \times \|\mathbf{L}_j(\boldsymbol{\alpha^*}_j)\|_1 + a_j(\boldsymbol{\alpha^*}_j) \quad \forall j \in [n] \\
&2 \times \|\mathbf{L}_{i_0}(\boldsymbol{\alpha^*}_{i_0})\|_1 - \|\mathbf{L}_{i_0}(\boldsymbol{\alpha^*}_{i_0}) + \mathbf{L}_{j_0}(\boldsymbol{\alpha^*}_{j_0})\|_1 > \frac{a_{i_0}(\boldsymbol{\alpha^*}_{i_0})}{\epsilon} - \frac{a_{j_0}(\boldsymbol{\alpha^*}_{j_0})}{\epsilon} \text{ for some } j_0 \in [n] \\
&\frac{1}{2} \times (-\epsilon \times (\|\mathbf{L}_{i_0}(\boldsymbol{\alpha^*}_{i_0}) + \mathbf{L}_{j_0}(\boldsymbol{\alpha^*}_{j_0})\|_1) + a_{i_0}(\boldsymbol{\alpha^*}_{i_0}) + a_{j_0}(\boldsymbol{\alpha^*}_{j_0})) > -\epsilon \times \|\mathbf{L}_{i_0}(\boldsymbol{\alpha^*}_{i_0})\|_1 + a_{i_0}(\boldsymbol{\alpha^*}_{i_0}) = t^*(\overline{G})
\end{aligned} \tag{24}
$$

$t^*(G) = \max_{\boldsymbol{\lambda}} G(\boldsymbol{\lambda})$ now consider $\overline{\boldsymbol{\lambda}} = (\boldsymbol{\alpha}^*_1, \ldots, \boldsymbol{\alpha}^*_m, \lambda_1 = 0, \ldots, \lambda_{i_0} = \frac{1}{2}, \ldots, \lambda_{j_0} = \frac{1}{2}, \ldots \lambda_n = 0)$

$$
\begin{aligned}
t^*(G) \geq G(\overline{\boldsymbol{\lambda}}) = \frac{1}{2} \times (-\epsilon \times (\|\mathbf{L}_{i_0}(\boldsymbol{\alpha^*}_{i_0}) + \mathbf{L}_{j_0}(\boldsymbol{\alpha^*}_{j_0})\|_1) + a_{i_0}(\boldsymbol{\alpha^*}_{i_0}) + a_{j_0}(\boldsymbol{\alpha^*}_{j_0})) \\
t^*(G) > -\epsilon \times \|\mathbf{L}_{i_0}(\boldsymbol{\alpha^*}_{i_0})\|_1 + a_{i_0}(\boldsymbol{\alpha^*}_{i_0}) = t^*(\overline{G}) \quad \text{Using Eq. 24}
\end{aligned}
$$

One simple example where this sufficient condition holds is $a_i(\boldsymbol{\alpha^*}_i) = a_j(\boldsymbol{\alpha^*}_j) = 0$ and $\mathbf{L}_{i_0}(\boldsymbol{\alpha^*}_{i_0}) = -\mathbf{L}_{j_0}(\boldsymbol{\alpha^*}_{j_0})$ and $-\mathbf{L}_{i_0}(\boldsymbol{\alpha^*}_{i_0})$ and $-\mathbf{L}_{j_0}(\boldsymbol{\alpha^*}_{j_0})$ are non-zero vectors.

D.2 Cross-execution bound refinement for conjunction of linear inequalities

We consider n executions of N on perturbed inputs given by $\{\mathbf{x_1} + \boldsymbol{\delta}, \ldots, \mathbf{x_n} + \boldsymbol{\delta}\}$. In this case, to prove the absence of **common** adversarial perturbation we need to show for all $i \in [n]$ the outputs $\mathbf{y_i} = N(\mathbf{x_i} + \boldsymbol{\delta})$ satisfy $\Psi(\mathbf{y_1}, \ldots, \mathbf{y_n}) = \bigvee_{i=1}^{n} \psi^i(\mathbf{y_i})$. Here, $\psi^i(\mathbf{y_i}) = \bigwedge_{j=1}^{m}(\mathbf{c_{i,j}}^T \mathbf{y_i} \geq 0)$ and $\mathbf{c_{i,j}} \in \mathbb{R}^{n_l}$. First, we prove lemmas necessary for characterizing the optimizable closed form that can be used for bound refinement.

Lemma 3. *$\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.((\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies \Psi(\mathbf{y_1}, \ldots, \mathbf{y_n}))$ if and only if $\left(\min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i}\right) \geq 0$ where for all $i \in [n]$, $\mathbf{y_i} = N(\mathbf{x_i} + \boldsymbol{\delta})$, $\Psi(\mathbf{y_1}, \ldots, \mathbf{y_n}) = \bigvee_{i=1}^{n} \psi^i(\mathbf{y_i})$ and $\psi^i(\mathbf{y_i}) = \bigwedge_{j=1}^{m}(\mathbf{c_{i,j}}^T \mathbf{y_i} \geq 0)$.*

Proof. We first show if $\left(\min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i}\right) \geq 0$ then $\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.((\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies \Psi(\mathbf{y_1}, \ldots, \mathbf{y_n}))$.

$$
\begin{aligned}
\left(\min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i}\right) \geq 0 &\implies (\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.(\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies (\max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i}) \geq 0) \\
&\implies (\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.(\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies \vee_{i=1}^{n}((\min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i}) \geq 0)) \\
&\implies \left(\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.(\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies \vee_{i=1}^{n} \wedge_{j=1}^{m} (\mathbf{c_{i,j}}^T \mathbf{y_i} \geq 0)\right) \\
&\implies (\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.((\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies \Psi(\mathbf{y_1}, \ldots, \mathbf{y_n})))
\end{aligned} \tag{25}
$$

Next, we show if $\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.((\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies \Psi(\mathbf{y_1}, \ldots, \mathbf{y_n}))$ then $\left(\min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i}\right) \geq 0$.

$$\begin{aligned}\left(\min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i}\right) < 0 &\implies (\exists \boldsymbol{\delta} \in \mathbb{R}^{n_0}.(\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \wedge ((\max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i}) < 0)) \\ &\implies (\exists \boldsymbol{\delta} \in \mathbb{R}^{n_0}.(\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \wedge \neg(\vee_{i=1}^{n} \psi^i(\mathbf{y_i}))) \\ &\implies \neg(\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.(\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies \Psi(\mathbf{y_1}, \ldots, \mathbf{y_n})) \end{aligned} \tag{26}$$

Eq. 26 is equivalent to showing the following

$$(\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.((\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies \Psi(\mathbf{y_1}, \ldots, \mathbf{y_n}))) \implies \left(\min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i}\right) \geq 0$$

Lemma 4. $\min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i} = \min_{j_1 \in [m], \ldots, j_n \in [m]} S(j_1, \ldots, j_n)$ *where for all* $i \in [n]$ *and* $j_i \in [m]$ $S(j_1, \ldots, j_n)$ *is defined as* $S(j_1, \ldots, j_n) = \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \mathbf{c_{i,j_i}}^T \mathbf{y_i}$.

Proof. First, we show $\min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i} \leq \min_{j_1 \in [m], \ldots, j_n \in [m]} S(j_1, \ldots, j_n)$.

$\mathbf{c_{i,j_i}}^T \mathbf{y_i} \geq \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i} \quad \forall i \in [n]$ and $\forall j_i \in [m]$

$S(j_1, \ldots, j_n) = \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \mathbf{c_{i,j_i}}^T \mathbf{y_i} \geq \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i} \quad \forall j_1 \in [m], \ldots, j_n \in [m]$

$$\min_{j_1 \in [m], \ldots, j_n \in [m]} S(j_1, \ldots, j_n) \geq \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i} \tag{27}$$

Next, we show $\min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i} \geq \min_{j_1 \in [m], \ldots, j_n \in [m]} S(j_1, \ldots, j_n)$.
There exists $\boldsymbol{\delta}^* \in \mathbb{R}^{n_0}$ such that $\|\boldsymbol{\delta}^*\|_\infty \leq \epsilon$, $\mathbf{y_i}^* = N(\mathbf{x_i} + \boldsymbol{\delta}^*)$ and $\max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i}^* = \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i}$. Let, $j_i^* = \arg\min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i}^*$ then

$S(j_1^*, \ldots, j_n^*) = \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \mathbf{c_{i,j_i^*}}^T \mathbf{y_i}$

$S(j_1^*, \ldots, j_n^*) \leq \max_{1 \leq i \leq n} \mathbf{c_{i,j_i^*}}^T \mathbf{y_i}^* = \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i}^* \quad$ since $j_i^* = \arg\min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i}^*$

$$\min_{j_1 \in [m], \ldots, j_n \in [m]} S(j_1, \ldots, j_n) \leq S(j_1^*, \ldots, j_n^*) \leq \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i} \tag{28}$$

Combining Eq. 27 and Eq. 28 we show $\min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i} = \min_{j_1 \in [m], \ldots, j_n \in [m]} S(j_1, \ldots, j_n)$.

Theorem 10. $\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.((\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies \Psi(\mathbf{y_1}, \ldots, \mathbf{y_n}))$ *if and only if* $\left(\min_{j_1 \in [m], \ldots, j_n \in [m]} S(j_1, \ldots, j_n)\right) \geq 0$ *where for all* $i \in [n]$, $\mathbf{y_i} = N(\mathbf{x_i} + \boldsymbol{\delta})$, $\Psi(\mathbf{y_1}, \ldots, \mathbf{y_n}) = \bigvee_{i=1}^{n} \psi^i(\mathbf{y_i})$, $\psi^i(\mathbf{y_i}) = \bigwedge_{j=1}^{m} (\mathbf{c_{i,j}}^T \mathbf{y_i} \geq 0)$ *and* $S(j_1, \ldots, j_n) = \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \mathbf{c_{i,j_i}}^T \mathbf{y_i}$.

Proof. Follows from lemma 3 and lemma 4.

Reduction to bound refinement with single linear inequality Theorem 10 allows us to learn parameters for each $S(j_1, \ldots, j_n)$ separately so that $S(j_1, \ldots, j_n) \geq 0$ for each $(j_1, \ldots, j_n)$ where each $j_i \in [m]$. For $S(j_1, \ldots, j_n)$, let $\{(\mathbf{L}_{j_1}(\boldsymbol{\alpha}_{j_1}), \mathbf{b}_{j_1}(\boldsymbol{\alpha}_{j_1})), \ldots, (\mathbf{L}_{j_n}(\boldsymbol{\alpha}_{j_n}), \mathbf{b}_{j_n}(\boldsymbol{\alpha}_{j_n}))\}$ denote the linear approximations satisfying $\mathbf{L}_{j_i}(\boldsymbol{\alpha}_{j_i})^T(\mathbf{x_i} + \boldsymbol{\delta}) + \mathbf{b}_{j_i}(\boldsymbol{\alpha}_{j_i}) \leq \mathbf{c_{i,j_i}}^T\mathbf{y_i}$ for any $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ such that $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$ and $\mathbf{l_{j_i}} \preceq \boldsymbol{\alpha}_{j_i} \preceq \mathbf{u_{j_i}}$. Then we can use cross-execution bound refinement for n executions to learn the parameters $(\boldsymbol{\alpha}_{j_1}, \ldots, \boldsymbol{\alpha}_{j_n})$. We repeat this process for all $(j_1, \ldots, j_n)$. However, the number of possible choices for $(j_1, \ldots, j_n)$ is m^n and learning parameters $(\boldsymbol{\alpha}_{j_1}, \ldots, \boldsymbol{\alpha}_{j_n})$ for all possible $(j_1, \ldots, j_n)$ is only practically feasible when both (m, n) are small constants. For larger values of (m, n) we greedily pick $(j_1, \ldots, j_n)$ for learning parameters to avoid the exponential blowup as detailed below.

Avoiding exponential blowup: Instead of learning parameters for all possible $(j_1, \ldots, j_n)$ we greedily select only single tuple $(j_1^*, \ldots, j_n^*)$. For the i-th execution with $\psi^i(\mathbf{y_i}) = \wedge_{i=1}^m(\mathbf{c_{i,j}}^T\mathbf{y_i} \geq 0)$, let $\{(\mathbf{L}_{i,1}(\boldsymbol{\alpha}_{i,1}^0), \mathbf{b}_{i,1}(\boldsymbol{\alpha}_{i,1}^0)), \ldots, (\mathbf{L}_{i,m}(\boldsymbol{\alpha}_{i,m}^0), \mathbf{b}_{i,m}(\boldsymbol{\alpha}_{i,m}^0))\}$ dentoes linear approximations satisfying $\mathbf{L}_{i,j}(\boldsymbol{\alpha}_{i,j}^0)^T(\mathbf{x_i} + \boldsymbol{\delta}) + \mathbf{b}_{i,j}(\boldsymbol{\alpha}_{i,j}^0) \leq \mathbf{c_{i,j}}^T\mathbf{y_i}$ for all $j \in [m]$ and for all $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ and $\|\boldsymbol{\delta}\| \leq \epsilon$. Note that for all $j \in [m]$, $\mathbf{l_i} \preceq \boldsymbol{\alpha}_{i,j}^0 \preceq \mathbf{u_i}$ are the initial values of the parameters $\boldsymbol{\alpha}_{i,j}$. Now, for we select j_i^* for each execution as $j_i^* = \arg\min_{j \in [m]} \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, \|\boldsymbol{\delta}\|_\infty \leq \epsilon} \mathbf{L}_{i,j}(\boldsymbol{\alpha}_{i,j}^0)^T(\mathbf{x_i} + \boldsymbol{\delta}) + \mathbf{b}_{i,j}(\boldsymbol{\alpha}_{i,j}^0)$.

Intuitively, we use j_i^* to determine the linear inequality $\mathbf{c_{i,j_i^*}}^T y_i \geq 0$ that is likely to be violated. For the tuple $(j_1^*, \ldots, j_n^*)$, let $\boldsymbol{\lambda}_{appx}^* = (\boldsymbol{\alpha}_{j_1^*}^*, \ldots, \boldsymbol{\alpha}_{j_n^*}^*, \lambda_{j_1^*}^*, \ldots, \lambda_{j_n^*}^*)$ denote the learned parameters (which may not correspond to global optimum). Then we use the same parameters across all m linear approximations for the i-th execution i.e. $\{(\mathbf{L}_{i,1}(\boldsymbol{\alpha^*}_{j_i^*}), \mathbf{b}_{i,1}(\boldsymbol{\alpha^*}_{j_i^*})), \ldots, (\mathbf{L}_{i,m}(\boldsymbol{\alpha^*}_{j_i^*}), \mathbf{b}_{i,m}(\boldsymbol{\alpha^*}_{j_i^*}))\}$. In this case, $t_{appx}^*(G)$ is defined as $t_{appx}^*(G) = \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{L}_{i,j}(\boldsymbol{\alpha^*}_{j_i^*})^T(\mathbf{x_i} + \boldsymbol{\delta}) + \mathbf{b}_{i,j}(\boldsymbol{\alpha^*}_{j_i^*})$. Next, we prove the correctness of the bound refinement.

Theorem 11 (Correctness of bound refinement for a conjunction of linear inequalities). *If* $t_{appx}^*(G) \geq 0$ *then* $\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.((\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies \Psi(\mathbf{y_1}, \ldots, \mathbf{y_n}))$ *where* $t_{appx}^*(G) = \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{L}_{i,j}(\boldsymbol{\alpha^*}_{j_i^*})^T(\mathbf{x_i} + \boldsymbol{\delta}) + \mathbf{b}_{i,j}(\boldsymbol{\alpha^*}_{j_i^*})$ *and for all* $i \in [n]$, $\mathbf{y_i} = N(\mathbf{x_i} + \boldsymbol{\delta})$.

Proof. First we show that $t_{appx}^*(G) \leq \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T\mathbf{y_i}$

$$\mathbf{L}_{i,j}(\boldsymbol{\alpha^*}_{j_i^*})^T(\mathbf{x_i} + \boldsymbol{\delta}) + \mathbf{b}_{i,j}(\boldsymbol{\alpha^*}_{j_i^*}) \leq \mathbf{c_{i,j}}^T\mathbf{y_i} \quad \forall i \in [n], \forall j \in [m] \text{ and for all } \boldsymbol{\delta} \in \mathbb{R}^{n_0} \text{ s.t } \|\boldsymbol{\delta}\|_\infty \leq \epsilon$$

$$\min_{1 \leq j \leq m} \mathbf{L}_{i,j}(\boldsymbol{\alpha^*}_{j_i^*})^T(\mathbf{x_i} + \boldsymbol{\delta}) + \mathbf{b}_{i,j}(\boldsymbol{\alpha^*}_{j_i^*}) \leq \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T\mathbf{y_i} \quad \forall i \in [n] \text{ and for all } \boldsymbol{\delta} \in \mathbb{R}^{n_0} \text{ s.t } \|\boldsymbol{\delta}\|_\infty \leq \epsilon$$

$$\max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{L}_{i,j}(\boldsymbol{\alpha^*}_{j_i^*})^T(\mathbf{x_i} + \boldsymbol{\delta}) + \mathbf{b}_{i,j}(\boldsymbol{\alpha^*}_{j_i^*}) \leq \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T\mathbf{y_i} \quad \text{for all } \boldsymbol{\delta} \in \mathbb{R}^{n_0} \text{ s.t } \|\boldsymbol{\delta}\|_\infty \leq \epsilon$$

$$\min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{L}_{i,j}(\boldsymbol{\alpha^*}_{j_i^*})^T(\mathbf{x_i} + \boldsymbol{\delta}) + \mathbf{b}_{i,j}(\boldsymbol{\alpha^*}_{j_i^*}) \leq \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T\mathbf{y_i}$$

$$t_{appx}^*(G) \leq \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T\mathbf{y_i} \tag{29}$$

Using lemma 3 and Eq 29

$$(t^*_{appx}(G) \geq 0) \implies \left(\min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, (\|\boldsymbol{\delta}\|_\infty \leq \epsilon)} \max_{1 \leq i \leq n} \min_{1 \leq j \leq m} \mathbf{c_{i,j}}^T \mathbf{y_i} \right) \geq 0$$
$$\implies \forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}. ((\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies \Psi(\mathbf{y_1}, \ldots, \mathbf{y_n}))$$

D.3 Handling general $\|\cdot\|_p$ norm

For general $\|\cdot\|_p$ norm we can generalize the dual formulation $G(\boldsymbol{\lambda})$ in the following way. Since, $\|\boldsymbol{\delta}\|_p \leq \epsilon$ and $a_i(\boldsymbol{\alpha}_i) = \mathbf{L}_i(\boldsymbol{\alpha}_i)^T \mathbf{x_i} + \mathbf{b}_i(\boldsymbol{\alpha}_i)$ then

$$G(\boldsymbol{\lambda}) = \min_{\|\boldsymbol{\delta}\|_p \leq \epsilon} \sum_{i=1}^{n} \lambda_i \times \left(\mathbf{L}_i(\boldsymbol{\alpha}_i)^T (\mathbf{x_i} + \boldsymbol{\delta}) + \mathbf{b}_i(\boldsymbol{\alpha}_i) \right)$$

$$G(\boldsymbol{\lambda}) = \sum_{i=1}^{n} \lambda_i \times a_i(\boldsymbol{\alpha}_i) + \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0},\ \|\boldsymbol{\delta}\|_p \leq \epsilon} \sum_{i=1}^{n} \lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i)^T \boldsymbol{\delta}$$

$$G(\boldsymbol{\lambda}) = \sum_{i=1}^{n} \lambda_i \times a_i(\boldsymbol{\alpha}_i) - \epsilon \times \left\| \sum_{i=1}^{n} \lambda_i \times \mathbf{L}_i(\boldsymbol{\alpha}_i) \right\|_q \quad \text{Using Hölder's Inequality with } \frac{1}{q} = 1 - \frac{1}{p}$$

D.4 MILP formulations and correctness

In this section, we show the MILP formulations for the k-UAP and worst-case hamming distance verification and present the theoretical results corresponding to the correctness and efficacy of the MILP formulations.

Let $I = \{i \mid \text{non-relational verifier does not verify } (\phi^i, \psi^i)\}$ denotes the executions that remain unverified by the non-relational verifier. For all $i \in I$, $j \in [m]$ let $(\mathbf{L}^{k'}_{i,j}, b^{k'}_{i,j})$ denote the linear approximations satisfying $\mathbf{L}^{k'}_{i,j}(\mathbf{x_i} + \boldsymbol{\delta}) + b^{k'}_{i,j} \leq \mathbf{c_{i,j}}^T \mathbf{y_i}$ for all $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ and $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$ where $k' \leq \sum_{i=1}^{k_1} \binom{k_0}{i} + 1$ and $\mathbf{y_i} = N(\mathbf{x_i} + \boldsymbol{\delta})$. Note that each linear approximations $(\mathbf{L}^{k'}_{i,j}, b^{k'}_{i,j})$ are obtained by the non-relational verifier or by the cross-execution bound refinement.

MILP formulations MILP formulation for k-UAP:

$$\min \quad M$$

$$\|\boldsymbol{\delta}\|_\infty \leq \epsilon$$
$$\mathbf{L}^{k'}_{i,j}(\mathbf{x_i} + \boldsymbol{\delta}) + b^{k'}_{i,j} \leq o_{i,j} \quad \forall i \in I, \forall j \in [m]\ \forall k'$$
$$z_i = \left(\left(\min_{j \in [m]} o_{i,j} \right) \geq 0 \right) \quad \text{for all } i \in I\ z_i \in \{0, 1\}$$
$$\overline{k} = k - |I| \quad \text{[number of executions verified by non-relational verifier]}$$
$$M = \sum_{i \in I} z_i + \overline{k} \tag{30}$$

MILP formulation for worst-case hamming distance:

$$\begin{aligned}
&\max \;\; M \\
&\\
&\|\boldsymbol{\delta}\|_\infty \leq \epsilon \\
&\mathbf{L}_{i,j}^{k'}(\mathbf{x_i} + \boldsymbol{\delta}) + b_{i,j}^{k'} \leq o_{i,j} \quad \forall i \in I,\, \forall j \in [m]\; \forall k' \\
&z_i = \left(\left(\min_{j \in [m]} o_{i,j} \right) \geq 0 \right) \quad \text{for all } i \in I \;\; z_i \in \{0,1\} \\
&M = |I| - \sum_{i \in I} z_i
\end{aligned}$$

Correctness for eliminating individually verified executions: We formally prove that eliminating individually verified executions is correct and does not lead to precision loss.

Theorem 12. $\mathbf{M}_0(\Phi,\Psi) = (k - |I|) + \min\limits_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, \|\boldsymbol{\delta}\|_\infty \leq \epsilon} \sum\limits_{i \in I} z_i(\boldsymbol{\delta})$ *where* $z_i(\boldsymbol{\delta})$ *is defined in Eq. 11,* $\mathbf{M}_0(\Phi,\Psi)$ *is defined in Eq. 13 and for all* $j \in [k] \setminus I$, $\forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.(\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies (z_j(\delta) = 1)$ *holds.*

Proof.

$$\begin{aligned}
\mathbf{M}_0(\Phi,\Psi) &= \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, \|\boldsymbol{\delta}\|_\infty \leq \epsilon} \sum_{i=1}^{k} z_i(\boldsymbol{\delta}) \\
&= \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, \|\boldsymbol{\delta}\|_\infty \leq \epsilon} \sum_{i \in ([k] \setminus I)} z_i(\boldsymbol{\delta}) + \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, \|\boldsymbol{\delta}\|_\infty \leq \epsilon} \sum_{i \in I} z_i(\boldsymbol{\delta}) \\
&= (k - |I|) + \min_{\boldsymbol{\delta} \in \mathbb{R}^{n_0}, \|\boldsymbol{\delta}\|_\infty \leq \epsilon} \sum_{i \in I} z_i(\boldsymbol{\delta}) \quad \text{since } \forall \boldsymbol{\delta} \in \mathbb{R}^{n_0}.(\|\boldsymbol{\delta}\|_\infty \leq \epsilon) \implies (z_j(\delta) = 1)
\end{aligned}$$

Soundness of MILP formulation: For soundness, we show that the optimal value of the MILP formulation (in Eq. 30) $\mathbf{M}(\Phi,\Psi)$ is always a valid lower bound of $\mathbf{M}_0(\Phi,\Psi)$. The soundness of the worst-case hamming distance formulation can be proved similarly.

Theorem 13 (Sondness of the MILP formulation in Eq. 30). $\mathbf{M}(\Phi,\Psi) \leq \mathbf{M}_0(\Phi,\Psi)$ *where* $\mathbf{M}(\Phi,\Psi)$ *is the optimal solution of the MILP in Eq. 30 and* $\mathbf{M}_0(\Phi,\Psi)$ *is defined in Eq. 13.*

Proof. We prove this by contradiction. Suppose, $\mathbf{M}(\Phi,\Psi) > \mathbf{M}_0(\Phi,\Psi)$ then there exists $\boldsymbol{\delta}^* \in \mathbb{R}^{n_0}$ such that $\|\boldsymbol{\delta}^*\|_\infty \leq \epsilon$ and $\mathbf{M}(\Phi,\Psi) > \mu(\boldsymbol{\delta}^*)$ where $\mu(\boldsymbol{\delta})$ defined in Eq. 12.

For all $i \in I$, $j \in [m]$ the linear approximation $(\mathbf{L}_{i,j}^{k'}, b_{i,j}^{k'})$ satisfies $\mathbf{L}_{i,j}^{k'}(\mathbf{x_i} + \boldsymbol{\delta}) + b_{i,j}^{k'} \leq \mathbf{c_{i,j}}^T \mathbf{y_i}$ for all $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ and $\|\boldsymbol{\delta}\|_\infty \leq \epsilon$ where $k' \leq \sum_{i=1}^{k_1} \binom{k_0}{i} + 1$ and $\mathbf{y_i} = N(\mathbf{x_i} + \boldsymbol{\delta})$. Let, $z_i^*(\boldsymbol{\delta}^*) = \left(\min\limits_{j \in [m]} o_{i,j}^*(\boldsymbol{\delta}^*) \geq 0 \right)$ where $o_{i,j}^*(\boldsymbol{\delta}^*) = \max\limits_{k'} \mathbf{L}_{i,j}^{k'}(\mathbf{x_i} +$

$\boldsymbol{\delta}^*) + b^{k'}_{i,j}$. Then $\mathbf{M}(\Phi,\Psi) \leq \overline{k} + \sum_{i\in I} z^*_i(\boldsymbol{\delta}^*)$ and $\mu(\boldsymbol{\delta}^*) < \overline{k} + \sum_{i\in I} z^*_i(\boldsymbol{\delta}^*)$.

$$\begin{aligned}
&\mu(\boldsymbol{\delta}^*) < \overline{k} + \sum_{i\in I} z^*_i(\boldsymbol{\delta}^*) \\
&\implies \sum_{i\in I} z_i(\boldsymbol{\delta}^*) < \sum_{i\in I} z^*_i(\boldsymbol{\delta}^*) \quad \text{where } z_i(\boldsymbol{\delta}^*) \text{ defined in Eq. 11} \qquad (31)
\end{aligned}$$

Eq. 31 implies that there exist $i_0 \in I$ such that $z_{i_0}(\boldsymbol{\delta}^*) = 0$ and $z^*_{i_0}(\boldsymbol{\delta}^*) = 1$. Since $z_{i_0}(\boldsymbol{\delta}^*) = 0$ then there exists $j_0 \in [m]$ such that $\mathbf{c_{i_0,j_0}}^T \mathbf{y_{i_0}}^* < 0$ where $\mathbf{y_{i_0}}^* = N(\mathbf{x_{i_0}} + \boldsymbol{\delta}^*)$

$$\begin{aligned}
&\min_{j\in[m]} o^*_{i_0,j}(\boldsymbol{\delta}^*) \leq o^*_{i_0,j_0}(\boldsymbol{\delta}^*) \leq \mathbf{c_{i_0,j_0}}^T \mathbf{y_{i_0}}^* < 0 \\
&(\min_{j\in[m]} o^*_{i_0,j}(\boldsymbol{\delta}^*) < 0) \implies (z^*_{i_0}(\boldsymbol{\delta}^*) = 0) \quad \text{Contradiction since } z^*_{i_0}(\boldsymbol{\delta}^*) = 1
\end{aligned}$$

Next, we show that RACoon is always at least as precise as the current SOTA relational verifier [65]. Note that [65] uses the same MILP formulation (Eq. 30) except instead of using k' linear approximations $\{(\mathbf{L}^1_{i,j}, b^1_{i,j}), \ldots, (\mathbf{L}^{k'}_{i,j}, b^{k'}_{i,j})\}$ it uses a single statically obtained linear approximation say $\{(\mathbf{L}^1_{i,j}, b^1_{i,j})\}$.

Theorem 14 (RACoon is at least as precise as [65]). $\mathbf{M}_b(\Phi,\Psi) \leq \mathbf{M}(\Phi,\Psi)$ *where* $\mathbf{M}(\Phi,\Psi)$ *is the optimal solution of the MILP in Eq. 30 and* $\mathbf{M}_b(\Phi,\Psi)$ *is the optimal solution from the baseline [65].*

Proof. Now we show that for $i \in I$, $\forall j \in [m]$ for every feasible value of the variable $o_{i,j}$ in Eq. 30 is also a feasible value of the same variable $o_{i,j}$ in MILP of [65]. Given $\forall k'$, $\mathbf{L}^{k'}_{i,j}(\mathbf{x_i} + \boldsymbol{\delta}) + b^{k'}_{i,j} \leq o_{i,j}$ then trivally $o_{i,j}$ satisfies condition $\mathbf{L}^1_{i,j}(\mathbf{x_i} + \boldsymbol{\delta}) + b^1_{i,j} \leq o_{i,j}$ used by the baseline [65]. Subsequently for all $i \in I$ every feasible value of z_i in Eq. 30 is also a feasible value of the same variable z_i in the MILP of [65]. Let. for all $i \in I$, $\mathcal{Z}$ and $\mathcal{Z}_b$ denote the sets of all feasible values of variables $(z_1, \ldots, z_I)$ from the MILP in Eq. 30 and the baseline [65] respectively. Then $\mathcal{Z} \subseteq \mathcal{Z}_b$ which implies

$$\begin{aligned}
\mathbf{M}_b(\Phi,\Psi) &\leq k - |I| + \min_{(z_1,\ldots,z_I)\in\mathcal{Z}_b} \sum_{i\in I} z_i \\
&\leq k - |I| + \min_{(z_1,\ldots,z_I)\in\mathcal{Z}} \sum_{i\in I} z_i = \mathbf{M}(\Phi,\Psi) \quad \text{Since } \mathcal{Z} \subseteq \mathcal{Z}_b
\end{aligned}$$

E Worst-case time complexity analysis of RACoon

Let, the total number of neurons in N be n_t and the number of layers in N is l. Then for each execution, the worst-case cost of running the non-relational verifier [61] is $O(l^2 \times n_t^3)$. We assume that we run I_t number of iterations with the optimizer and the cost of each optimization step over a set of n executions is $O(n \times C_o)$. In general, C_o is similar to the cost of the non-relational verifier i.e.

$O(l^2 \times n_t^3)$. Then the total cost of cross-execution refinement is $O(T \times I_t \times C_o)$ where $T = \sum_{i=1}^{k_1} \left(\binom{k_0}{i} \times i \right)$. Assuming MILP with $O(k \times n_l)$ integer variables in the worst-case takes $C_M(k \times n_l)$ time. Then the worst-case complexity of RACoon is $O(k \times l^2 \times n_t^3) + O(T \times I_t \times C_o) + C_M(k \times n_l)$.

F Details of DNN archietectures

Table 3: DNN architecture details

Dataset	Model	Type	Train	# Layers	# Params
MNIST	IBPSmall	Conv	IBP	4	60k
	ConvSmall	Conv	Standard	4	80k
	ConvSmall	Conv	PGD	4	80k
	ConvSmall	Conv	DiffAI	4	80k
	ConvSmall	Conv	COLT	4	80k
	IBPMedium	Conv	IBP	5	400k
	ConvBig	Conv	DiffAI	7	1.8M
CIFAR10	IBP-Small	Conv	IBP	4	60k
	ConvSmall	Conv	Standard	4	80k
	ConvSmall	Conv	PGD	4	80k
	ConvSmall	Conv	DiffAI	4	80k
	ConvSmall	Conv	COLT	4	80k
	IBPMedium	Conv	IBP	5	2.2M
	ConvBig	Conv	DiffAI	7	2.5M

F.1 Implementation Details

We implemented our method in Python with Pytorch V1.11 and used Gurobi V10.0.3 as an off-the-shelf MILP solver. The implementation of cross-execution bound refinement is built on top of the SOTA DNN verification tool auto_LiRPA [62] and uses Adam [24] for parameter learning. We run 20 iterations of Adam on each set of executions. For each relational property, we use $k_0 = 6$ and $k_1 = 4$ for deciding which set of executions to consider for cross-execution refinement as discussed in section 3.2. 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.

G Proofs

In this section, we provide formal proofs for theorems in Sections 5 and 5.

G.1 Proofs for k-Common Perturbation Bounding

In this section, we provide a formal proof for Theorem 3.

Lemma 5. *Given* $(\mathbf{x}, y) \in \mathcal{X}$*, network* $f : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$*, norm-bound* $\epsilon \in \mathbb{R}$*,* $\mathbf{u}^*$ *as defined in Eq. 7, and* $k \in \mathbb{N}$ *s.t.* $\Psi(\mathbf{u}^*, k)$*, then*

$$\mathcal{L}(f(\mathbf{x}+\mathbf{u}^*), y) \leq \max_{\mathbf{u} \in \mathcal{C}_{\mathcal{X},f}(k,\epsilon)} \mathcal{L}(f(\mathbf{x}+\mathbf{u}), y)$$

Proof. $\mathbf{u}^* \in \mathcal{B}(\mathbf{0}, \epsilon)$ by definition of $\mathbf{u}^*$. $\Psi(\mathbf{u}^*, k)$ by definition of k. Therefore, $\mathbf{u}^* \in \mathcal{C}_{\mathcal{X},f}(k, \epsilon)$. The statement of the lemma then follows by definition of max as $\forall i \in [j].x_i \leq \max(x_1, \ldots, x_j)$.

Lemma 6. *Given* $(\mathbf{x}, y) \in \mathcal{X}$*, network* $f : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$*, norm-bound* $\epsilon \in \mathbb{R}$*,* $\mathbf{u}^*$ *as defined in Eq. 7, and* $k, j \in \mathbb{N}$ *s.t.* $k < j$*, then*

$$\max_{\mathbf{u} \in \mathcal{C}_{\mathcal{X},f}(j,\epsilon)} \mathcal{L}(f(\mathbf{x}+\mathbf{u}), y) \leq \max_{\mathbf{u} \in \mathcal{C}_{\mathcal{X},f}(k,\epsilon)} \mathcal{L}(f(\mathbf{x}+\mathbf{u}), y)$$

Proof. If $\mathbf{u} \in \mathcal{C}_{\mathcal{X},f}(j, \epsilon)$ then we have $\mathbf{u} \in \mathcal{B}(\mathbf{0}, \epsilon)$ and $\Psi(\mathbf{u}, j) = 1$. $\forall i < j.\Psi(\mathbf{u}, j) \implies \Psi(\mathbf{u}, i)$ as the existence of a tuple of size j which is all misclassified by u means that all subsets of that tuple (sizes $i < j$) are misclassified by u. Therefore, $\forall k < j.\mathbf{u} \in \mathcal{C}_{\mathcal{X},f}(j, \epsilon) \implies \mathbf{u} \in \mathcal{C}_{\mathcal{X},f}(k, \epsilon)$ and the statement of the lemma follows through the definition of max.

Theorem 15. *Given* $\mathcal{X} \subseteq \mathbb{R}^{d_{in}} \times \mathbb{N}$*, network* $f : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$*,* $\mathbf{u}^*$ *as defined in Equation* (7)*, and norm-bound* $\epsilon \in \mathbb{R}$*. Let* $\kappa^* = \hat{\Psi}_{\mathcal{X},f}(\mathbf{u}^*)$ *and*

$$\mathcal{E}(k, \epsilon) = \mathop{\mathbb{E}}_{(x,y)\in\mathcal{X}} \left[\max_{\mathbf{u} \in \mathcal{C}_{\mathcal{X},f}(k,\epsilon)} \mathcal{L}(f(\mathbf{x}+\mathbf{u}), y) \right]$$

then,

$$\max_{\mathbf{u}\in\mathcal{B}(0,\epsilon)} \left(\mathop{\mathbb{E}}_{(x,y)\in\mathcal{X}} [\mathcal{L}(f(\mathbf{x}+\mathbf{u}), y)] \right) \leq \mathcal{E}(\kappa^*, \epsilon) \leq \mathcal{E}(\kappa^* - 1, \epsilon) \leq \cdots \leq \mathcal{E}(1, \epsilon)$$

Proof.

$$\begin{aligned}
\max_{\mathbf{u}\in\mathcal{B}(0,\epsilon)} \left(\mathop{\mathbb{E}}_{(x,y)\in\mathcal{X}} [\mathcal{L}(f(\mathbf{x}+\mathbf{u}), y)] \right) &= \mathop{\mathbb{E}}_{(x,y)\in\mathcal{X}} [\mathcal{L}(f(\mathbf{x}+\mathbf{u}^*), y)] && \text{(by Eq. 7)} \\
&\leq \mathop{\mathbb{E}}_{(x,y)\in\mathcal{X}} \left[\max_{\mathbf{u} \in \mathcal{C}_{\mathcal{X},f}(\kappa^*,\epsilon)} \mathcal{L}(f(\mathbf{x}+\mathbf{u}), y) \right] && \text{(by Lm. 5)} \\
&= \mathcal{E}(\kappa^*, \epsilon) && \text{(by Def)} \\
&\leq \mathcal{E}(\kappa^* - 1, \epsilon) && \text{(by Lm. 6)} \\
&\cdots \\
&\leq \mathcal{E}(1, \epsilon) && \text{(by Lm. 6)}
\end{aligned}$$

G.2 Proofs for $\mathcal{L}_{\text{CITRUS}}$ being an upper bound for $\mathcal{L}_{\text{2CP}}$

In this section, we provide a formal proof for Theorem 4. For these proofs, we assume a standard loss function where additive perturbations which are adversarial incur greater loss than additive perturbations which are safe, i.e. $\forall \mathbf{v}, \mathbf{v}' \in \mathcal{B}(\mathbf{0}, \epsilon). \neg A_f(\mathbf{x}_0 + \mathbf{v}, y) \wedge A_f(\mathbf{x}_0 + \mathbf{v}', y) \implies \mathcal{L}(f(\mathbf{x}_0 + \mathbf{v}), y_0) \leq \mathcal{L}(f(\mathbf{x}_0 + \mathbf{v}'), y_0)$.

Definition 5. *The **adversarial set**, $\mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon) \subseteq \mathcal{B}(\mathbf{0}, \epsilon)$, for a point $(\mathbf{x}_0, y_0)$ is defined as the set of points in $\mathcal{B}(\mathbf{0}, \epsilon)$ which cause f to misclassify when added to $\mathbf{x}_0$. That is,*

$$\mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon) \coloneqq \{\mathbf{v} | A_f(\mathbf{x}_0 + \mathbf{v}, y_0) \wedge \mathbf{v} \in \mathcal{B}(\mathbf{0}, \epsilon)\}$$

Further, let $\neg\mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon) \subseteq \mathcal{B}(\mathbf{0}, \epsilon)$ indicate the safe set. That is,

$$\neg\mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon) \coloneqq \{\mathbf{v} | \neg A_f(\mathbf{x}_0 + \mathbf{v}, y_0) \wedge \mathbf{v} \in \mathcal{B}(\mathbf{0}, \epsilon)\}$$

Using the definition of an adversarial set, we can now show that the loss for $(\mathbf{x}_0, y_0)$ over $\mathcal{C}_{\mathcal{X}_B, f}(2, \epsilon)$ must occur in the adversarial set for $(\mathbf{x}_0, y_0)$.

Lemma 7. *Given $\mathcal{X}_B \subseteq \mathbb{R}^{d_{in}} \times \mathbb{Z}$, a network $f : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$, a given input $(\mathbf{x}_0, y_0)$, and norm-bound $\epsilon \in \mathbb{R}$. If $\mathcal{C}_{\mathcal{X}_B, f}(2, \epsilon) \cap \mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon) \neq \emptyset$ then,*

$$\mathcal{L}_{2CP}(\mathcal{X}_B, \mathbf{x}_0, y_0, \epsilon) = \max_{\mathbf{u} \in \mathcal{C}_{\mathcal{X}_B, f}(2, \epsilon) \cap \mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon)} \mathcal{L}(f(\mathbf{x}_0 + \mathbf{u}), y_0)$$

Proof. By definition we have that,

$$\mathcal{L}_{\text{2CP}}(\mathcal{X}_B, \mathbf{x}_0, y_0, \epsilon) = \max_{\mathbf{u} \in \mathcal{C}_{\mathcal{X}_B, f}(2, \epsilon)} \mathcal{L}(f(\mathbf{x} + \mathbf{u}), y)$$

Let $\mathbf{u}'$ be the point which maximizes the RHS of the definition above (there may be multiple points which maximize the RHS; however, without loss of generality assume that $\mathbf{u}'$ is unique), that is

$$\max_{\mathbf{u} \in \mathcal{C}_{\mathcal{X}_B, f}(2, \epsilon)} \mathcal{L}(f(\mathbf{x} + \mathbf{u}), y) = \mathcal{L}(f(\mathbf{x} + \mathbf{u}'), y)$$

By the assumption we made for standard loss functions, we have that

$$\max_{\mathbf{u} \in \neg\mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon)} \mathcal{L}(f(\mathbf{x} + \mathbf{u}), y) \leq \max_{\mathbf{u} \in \mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon)} \mathcal{L}(f(\mathbf{x} + \mathbf{u}), y)$$

This implies that

$$\max_{\mathbf{u} \in \mathcal{C}_{\mathcal{X}_B, f}(2, \epsilon) \cap \neg\mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon)} \mathcal{L}(f(\mathbf{x} + \mathbf{u}), y) \leq \max_{\mathbf{u} \in \mathcal{C}_{\mathcal{X}_B, f}(2, \epsilon) \cap \mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon)} \mathcal{L}(f(\mathbf{x} + \mathbf{u}), y)$$

Therefore, since $\mathbf{u}'$ maximizes the loss over $C_{\mathcal{X}_B, f}(2, \epsilon)$ the above inequality implies that $\mathbf{u}' \in C_{\mathcal{X}_B, f}(2, \epsilon) \cap \mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon)$. That is,

$$\mathcal{L}(f(\mathbf{x} + \mathbf{u}'), y) = \max_{\mathbf{u} \in \mathcal{C}_{\mathcal{X}_B, f}(2, \epsilon) \cap \mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon)} \mathcal{L}(f(\mathbf{x} + \mathbf{u}), y)$$

Which gives the statement of the lemma with the definition of $\mathbf{u}'$.

Definition 6. *We define the* ***cross-input adversarial set****,* $\mathcal{CI}_f(\mathcal{X}_B, \mathbf{x}_0, y_0, \epsilon)$*, to be the intersection between the adversarial sets for all points in* $\mathcal{X}_B$ *besides* $(\mathbf{x}_0, y_0)$*. That is,*

$$\mathcal{CI}_f(\mathcal{X}_B, \mathbf{x}_0, y_0, \epsilon) \coloneqq \bigcup_{(\mathbf{x}_i, y_i) \in \mathcal{X}_B, \mathbf{x}_i \neq \mathbf{x}_0} \mathcal{S}_f(\mathbf{x}_i, y_i, \epsilon)$$

Note that, $\mathcal{CI}_f(\mathcal{X}_B, \mathbf{x}_0, y_0, \epsilon) = \mathcal{C}_{\mathcal{X}_B/(\mathbf{x}_0, y_0), f}(1, \epsilon)$.

Using the definition of $\mathcal{CI}_f(\mathcal{X}_B, \mathbf{x}_0, y_0, \epsilon)$ we can bound the maximum loss for the current input, $(\mathbf{x}_0, y_0)$, over the intersection of $\mathcal{C}_{\mathcal{X}_B, f}(2, \epsilon)$ and $\mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon)$ by maximizing the loss over the cross-input adversarial set for $(\mathbf{x}_0, y_0)$.

Lemma 8. *Given* $\mathcal{X}_B \subseteq \mathbb{R}^{d_{in}} \times \mathbb{Z}$*, a network* $f : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$*, a given input* $(\mathbf{x}_0, y_0)$*, and norm-bound* $\epsilon \in \mathbb{R}$*. Then,*

$$\max_{\mathbf{u} \in \mathcal{C}_{\mathcal{X}_B, f}(2, \epsilon) \cap \mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon)} \mathcal{L}(f(\mathbf{x}_0 + \mathbf{u}), y_0) \leq \max_{\mathbf{u} \in \mathcal{CI}_f(\mathcal{X}_B, \mathbf{x}_0, y_0, \epsilon)} \mathcal{L}(f(\mathbf{x}_0 + \mathbf{u}), y_0)$$

Proof. Given $\mathbf{u} \in \mathcal{C}_{\mathcal{X}_B, f}(2, \epsilon) \cap \mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon)$ by definition of $\mathcal{C}_{\mathcal{X}_B, f}(2, \epsilon)$ we know that $\exists\{(\mathbf{x}_i, y_i), (\mathbf{x}_j, y_j)\} \subseteq \mathcal{X}_B . A_f(\mathbf{x}_i + \mathbf{u}, y_i) \wedge A_f(\mathbf{x}_j + \mathbf{u} + y_j)$. By definition of $\mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon)$ we know that one of $\mathbf{x}_i, \mathbf{x}_j$ equals $\mathbf{x}_0$. Thus, we know that there is at least one more input which $\mathbf{u}$ is adversarial for, that is $\exists(\mathbf{x}_i, y_i) \subseteq \mathcal{X}_B, \mathbf{x}_i \neq \mathbf{x_0} . A_f(\mathbf{x}_i + \mathbf{u}, y_i) \wedge A_f(\mathbf{x}_j + \mathbf{u} + y_j)$. This implies that $\exists(\mathbf{x}_i, y_i) \subseteq \mathcal{X}_B, \mathbf{x}_i \neq \mathbf{x_0} . \mathbf{u} \in \mathcal{S}_f(\mathbf{x}_i, y_i, \epsilon)$. If we take the union of all the adversarial sets of all other inputs, this union must also contain $\mathbf{u}$, in other words, $\mathbf{u} \in \mathcal{CI}_f(\mathcal{X}_B, \mathbf{x}_0, y_0, \epsilon)$. This gives us that,

$$\mathbf{u} \in \mathcal{C}_{\mathcal{X}_B, f}(2, \epsilon) \cap \mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon) \implies \mathbf{u} \in \mathcal{CI}_f(\mathcal{X}_B, \mathbf{x}_0, y_0, \epsilon)$$

The statement of the lemma follows by the definition of max.

Using Lemmas 7 and 8 we now have our main result which leads to CITRUS loss.

Theorem 16. *Given* $\mathcal{X}_B \subseteq \mathbb{R}^{d_{in}} \times \mathbb{Z}$*, a network* $f : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$*, a given input* $(\mathbf{x}_0, y_0)$*, and norm-bound* $\epsilon \in \mathbb{R}$*. Then,*

$$\mathcal{L}_{2CP}(\mathcal{X}_B, \mathbf{x}_0, y_0, \epsilon) \leq \max_{\mathbf{u} \in \mathcal{C}_{\mathcal{X}/(\mathbf{x}_0, y_0), f}(1, \epsilon)} \mathcal{L}(f(\mathbf{x}_0 + \mathbf{u}), y_0)$$

Proof. Case 1. $\mathcal{C}_{\mathcal{X}_B, f}(2, \epsilon) \cap \mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon) \neq \emptyset$

$$\begin{aligned}
\mathcal{L}_{\text{2CP}}(\mathcal{X}_B, \mathbf{x}_0, y_0, \epsilon) &= \max_{\mathbf{u} \in \mathcal{C}_{\mathcal{X}_B, f}(2, \epsilon) \cap \mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon)} \mathcal{L}(f(\mathbf{x}_0 + \mathbf{u}), y_0) && \text{(by Lemma 7)} \\
&\leq \max_{\mathbf{u} \in \mathcal{CI}_f(\mathcal{X}_B, \mathbf{x}_0, y_0, \epsilon)} \mathcal{L}(f(\mathbf{x}_0 + \mathbf{u}), y_0) && \text{(by Lemma 8)} \\
&= \max_{\mathbf{u} \in \mathcal{C}_{\mathcal{X}/(\mathbf{x}_0, y_0), f}(1, \epsilon)} \mathcal{L}(f(\mathbf{x}_0 + \mathbf{u}), y_0) && \text{(by Definition 6)}
\end{aligned}$$

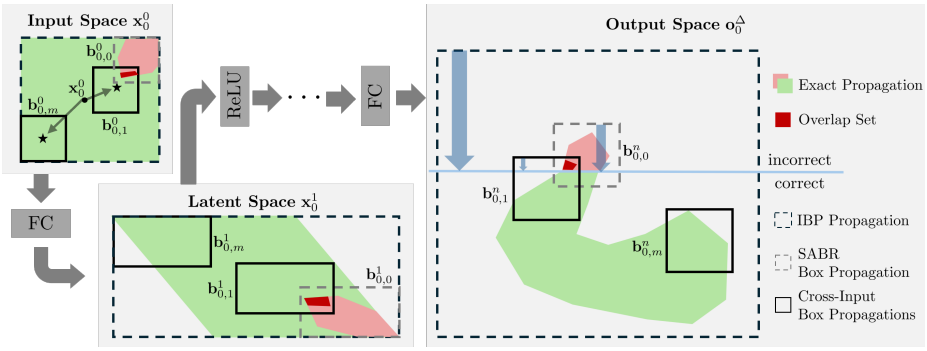


Fig. 3: Propagation for $\mathbf{x}_0$ and bounding boxes $b^0_{0,1}$ and $b^0_{0,m}$. The boxes are propagated using IBP through each layer of the network. In the output layer, we see that although $b^n_{0,1}$ does not capture the entire adversarial region for $\mathbf{x}_0$ it does capture the overlapping adversarial region between $\mathbf{x}_0$ and $\mathbf{x}_1$. Compared to IBP propagation (⬚) we induce much less regularization (big blue ⬇ vs. small blue ⬇). Depending on the size of the overlap region, SABR (⬚) incurs more regularization (medium blue ⬇).

Case 2. $\mathcal{C}_{\mathcal{X}_B,f}(2,\epsilon) \cap \mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon) = \emptyset$

$\mathcal{C}_{\mathcal{X}_B,f}(2,\epsilon) \cap \mathcal{S}_f(\mathbf{x}_0, y_0, \epsilon) = \emptyset$ implies $(\mathbf{x}_0, y_0)$ is not susceptible to any universal perturbation which affects it and another input at the same time, that is $\forall \mathbf{u} \in \mathcal{B}(\mathbf{0}, \epsilon), (\mathbf{x}_i, y_i) \in \mathcal{X}_B, \mathbf{x}_i \neq \mathbf{x_0}.\neg(A_f(\mathbf{x}_0 + \mathbf{u}, y_0) \wedge A_f(\mathbf{x}_i + \mathbf{u}, y_i))$. In other words, $(\mathbf{x}_0, y_0)$ is already safe from UAPs and training on adversarial sets for other inputs will not incur regularization.

H CITRUS Box Propagation

We illustrate the propagation process for one input, $\mathbf{x}_0$, in Figure 3. All boxes can be propagated through the network using any symbolic propagation method [51,63,19], but similarly to SABR we use Box propagation [34,19] for its speed and well-behaved optimization problem [22]. In the output space, we visualize the propagation of the cross-input adversarial boxes (☐) and the single-input adversarial, or SABR, box (⬚). The dark red region (■) represents the common perturbation set between $\mathbf{x}_0$ and $\mathbf{x}_1$. We can see that at the end of the propagation $b^n_{0,1}$ includes the entire overlap set even though it does not include the entire adversarial set for $\mathbf{x}_0$. Even though CITRUS propagates more boxes than SABR, many of these boxes incur no regularization ($b^n_{0,m}$) while other boxes include a smaller portion of the adversarial region ($b^n_{0,1}$) incurring less regularization (medium blue ⬇ vs. small blue ⬇) compared to SABR propagation (⬚). We also show IBP propagation of the entire l_∞ ball and we see that CITRUS incurs significantly less regularization (big blue ⬇).

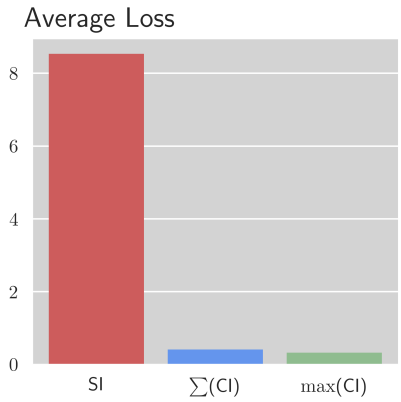
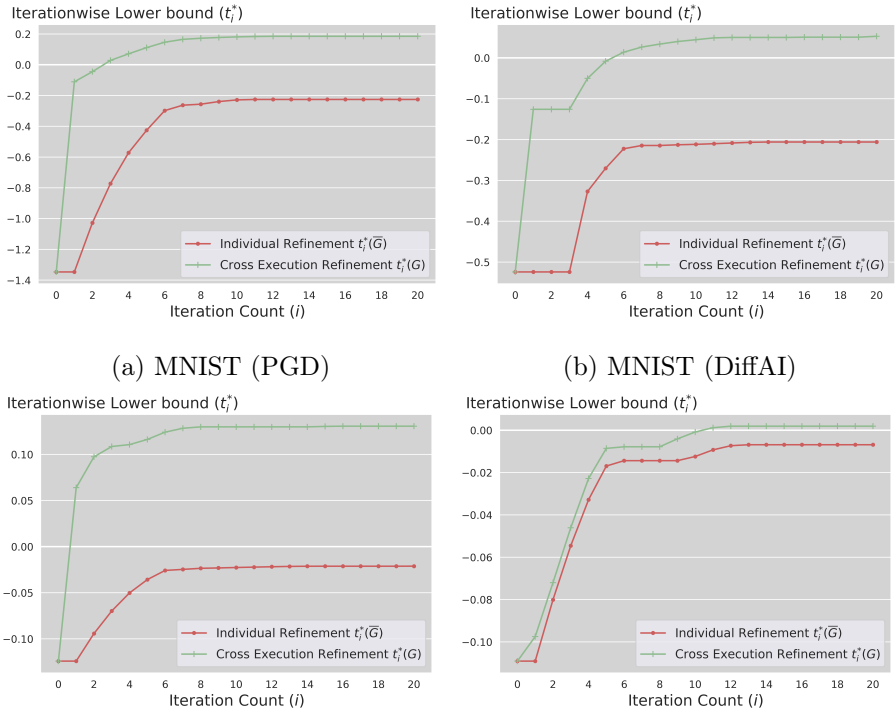


Fig. 4: Comparison of average loss across the last epoch of training incurred by same-input adversarial boxes to the sum and max of cross-input adversarial boxes. Network trained on CIFAR-10 with $\epsilon = \frac{8}{255}$ and batch size of 8.

I Same-Input vs. Cross-Input Adversarial Boxes

Figure 1 and 3 give us some insight into why we do not propagate the same-input adversarial box. To measure the impact of adding same-input adversarial boxes to the loss, we record the average loss across the last epoch of training for a convolutional network trained on CIFAR-10 with $\epsilon = 8/255$ and a batch size of 5. We compare the loss from the same-input adversarial box (SI) with the sum ($\sum$(CI)) and max loss (max(CI)) from the 4 cross-input adversarial boxes. In Figure 4 we see that the loss from the same-input adversarial box dominates the loss from the other boxes substantially. This indicates that adding same-input boxes likely incurs significant regularization and reduction in standard accuracy. This is corroborated by our ablation study in Appendix M where we find that adding the same-input adversarial boxes to the training process reduces the model's standard accuracy. We further observe that for the cross-input adversarial boxes, the max loss is close to the sum indicating that a single large loss typically dominates the sum.

J Cross-executional bound improvement



(a) MNIST (PGD)

(b) MNIST (DiffAI)

(c) CIFAR10 (PGD)

(d) CIFAR10 (DiffAI)

Fig. 5: Lower bound (t in Eq. 14) from individual vs. cross executional bound refinement over 2 executions on ConvSmall networks.

K CITRUS Further Experimental Setup Details

In this section, we provide details on experimental setup as well as runtimes.

K.1 Training and Architecture Details

We use a batch-size of 5 when training for all experiments in the paper. We use a similar setup to prior works in certified training [37,33,48]. Including the weight initialization and regularization from Shi *et al.* [48] and the τ/ϵ ratio from Müller *et al.* [37]. We used a longer PGD search with 20 steps (vs 8 for SABR) when selecting the centers for our propagation regions.

Similar to prior work [37,33,48] we use a 7-layer convolutional DNN, CNN7. The first 5 layers are convolutional with filter sizes of $[64, 64, 128, 128, 128]$, kernel size 3, strides $[1, 1, 2, 1, 1]$, and padding 1. Then followed by two fully connected layers of size 512 and one with size of the output dimension.

K.2 Training/Verification Runtimes

All experiments were performed on a desktop PC with a GeForce RTX(TM) 3090 GPU and a 16-core Intel(R) Core(TM) i9-9900KS CPU @ 4.00GHz. The training runtimes for different datasets and ϵs can be seen in Table 4. Training for times for CITRUS are roughly linear to batch-size and SABR runtimes, for example, SABR takes around 238 minutes to train on the same hardware for CIFAR-10 and $\epsilon = 8/255$ which is 4.21× less than CITRUS. Verification for CITRUS trained networks on the entire test dataset takes around 8h for MNIST, 11h for CIFAR-10, and 18h for TinyImageNet for each network.

Table 4: CITRUS training runtimes.

Dataset	ϵ	Time (mins)
MNIST	0.1	527
	0.3	492
CIFAR-10	2/255	1004
	8/255	1185
TinyImageNet	1/255	3583

L Comparison to Universal Adversarial Training

In Table 5 we compare the performance of CITRUS to universal and standard adversarial training methods. We compare to standard single-input PGD adversarial training [32]. Universal Adversarial Training (UAT) introduced by Shafahi *et al.* [47] introduces an efficient way to perform adversarial training against UAPs. Benz *et al.* [5] introduces a class-wise variant of the UAT algorithm which improves performance. We compare against all of these methods in Table 5 on CIFAR-10 with $\epsilon = 8/255$. We observe that while these methods obtain good standard accuracy (94.91% for CW-UAT compared to 63.12% for CITRUS) they perform poorly for certified average UAP accuracy (1.51% for CW-UAT compared to 39.88% for CITRUS). Our observation for the certified UAP accuracy of adversarial training is consistent with other studies which show that the standard certified accuracy of adversarial training based methods is low [34].

M CITRUS Ablation Studies

Propagation Region Size. To study the effect of different values of τ we vary the ratio of $\tau/\epsilon \in [0.3, 0.9]$ keeping all other training parameters constant. We perform our experiment on CIFAR-10 with $\epsilon = \frac{8}{255}$. In Figure 6, we see that increasing the ratio tends to decrease the final standard accuracy and increases the certified average UAP accuracy. Larger boxes capture more of the true adversarial region but they also incur more regularization.

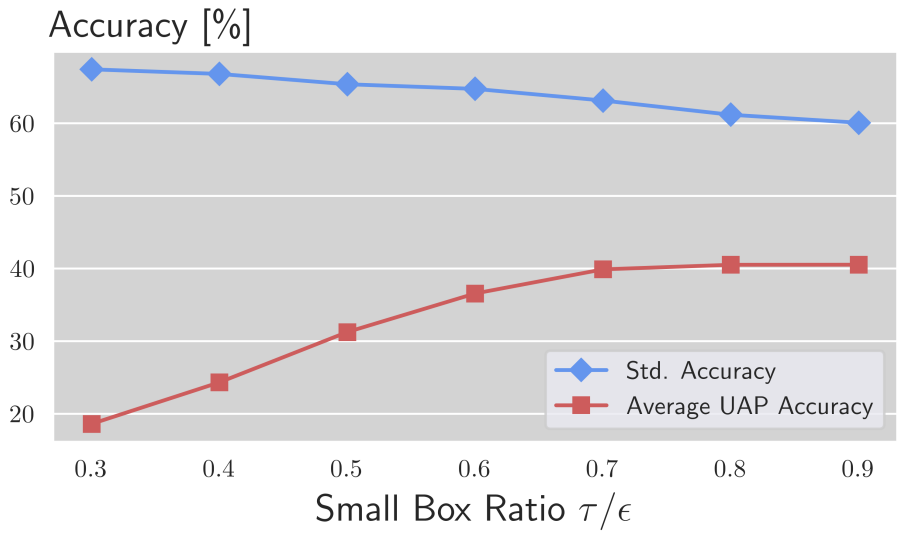


Fig. 6: Comparison of standard accuracy and certified average UAP accuracy across different ratios of τ/ϵ for CITRUS on CIFAR-10 with $\epsilon = \frac{8}{255}$.

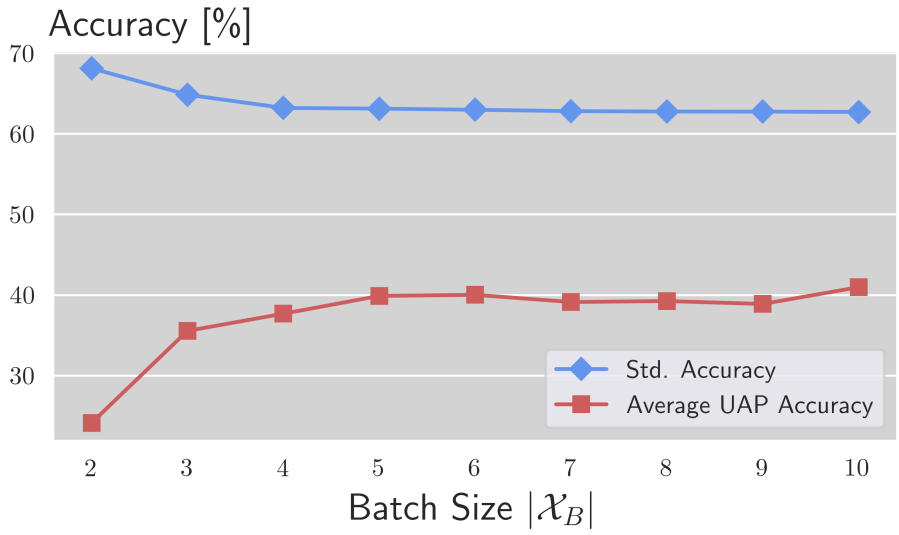


Fig. 7: Comparison of standard accuracy and certified average UAP accuracy across different batch sizes for CITRUS on CIFAR-10 with $\epsilon = \frac{8}{255}$.

Fig. 8: Ablation studies on values of τ and batch size.

Table 5: Comparison of standard accuracy (Std) and certified average UAP accuracy (UCert) for different universal and standard adversarial training methods on the full CIFAR-10 test sets. A variation on [64] is used for certified worst-case UAP accuracy.

Dataset	ϵ	Training Method	Source	Std [%]	UCert [%]
CIFAR-10	$\frac{8}{255}$	PGD	Madry *et al.* [32]	87.25	0.0
		UAT	Shafahi *et al.* [47]	94.28	0.87
		CW-UAT	Benz *et al.* [5]	**94.91**	1.51
		CITRUS	this work	63.12	**39.88**

Training Batch Size. CITRUS trains each input to be robust on the adversarial regions coming from other inputs in the batch. To study the effect of batch size on overall training performance, we vary the batch size from 2 to 10 keeping all other training parameters constant. We perform our experiment on CIFAR-10 with $\epsilon = \frac{8}{255}$. In Figure 7, we see that increasing the batch size tends to decrease the standard accuracy but increases the certified average UAP accuracy. A batch size of 2 means that each input only sees the adversarial region from 1 other input, so the certified average UAP accuracy is low. This quickly increases then stabilizes. We also note a linear relationship in runtime as more inputs per batch mean that we have to propagate more boxes through the network for each input.

Same-Input Adversarial Boxes. To compare the effect of adding same-input adversarial boxes to CITRUS training, visualized in Figure 1 c) compared to Figure 1 d), we train a network on CIFAR-10 with $\epsilon = 8/255$ using CITRUS but include same-input adversarial boxes while training. In Table 6, we observe that adding same-input adversarial boxes results in a severe reduction in standard accuracy and a slight increase in certified average UAP accuracy. CITRUS + SI has similar performance to SABR. Figure 4 shows the average loss coming from same-input adversarial boxes dominates the loss coming from cross-input adversarial boxes which explains why adding same-input adversarial boxes to CITRUS yields similar performance to SABR.

Table 6: Comparing the effect of adding same-input adversarial boxes (CITRUS + SI) to CITRUS on CIFAR-10 with $\epsilon = \frac{8}{255}$.

Method	Std [%]	UCert [%]
CITRUS	**63.12**	39.88
CITRUS + SI	50.24	**40.06**