The *Diff* MILP Model and PCA for *Real-Time* Verification of DNN Robustness

No Author Given

No Institute Given

Abstract. Deep neural networks (DNNs) are often brittle to small perturbations, which has led to extensive research to verify their robustness. While most existing methods focus on *local* robustness, i.e., verification in the neighborhood of fixed specific inputs, these techniques are often impractical for guaranteeing robustness to *real-time* inputs on embedded systems, due to excessive latency and computational cost.
To tackle real-time verification, we proceed in two steps. First, offline, we solve *global* robustness problems, which are significantly more complex than local robustness. We then use the results online to perform real-time verification. Offline, we compute *bounds* on the difference of output values across different DNN decision classes under both L_∞- *and* L_1-perturbations. To enable this, we develop novel *Diff* MILP models, explicitly representing the small *diff*erential variables between the input and its perturbation, instead of implicitly representing them as differences between 2 larger values (standard MILP encoding). To further improve the bounds, we employ principal component analysis (PCA) to focus on *realistic* in-distribution inputs. Online, our method verifies in real-time over 70% of incoming images across standard benchmarks (MNIST, Fashion-MNIST, and CIFAR-10 with L_1-perturbations of $1, 4, 2$, respectively), requiring only 0.5 ms of latency on a single CPU core.

1 Introduction

While deep neural networks (DNNs) have demonstrated remarkable capabilities, achieving human-like or even superior performance across a wide range of tasks, their robustness is often compromised by their susceptibility to input perturbations [25]. This vulnerability has catalyzed the formal verification community to develop various methodologies, each presenting a unique trade-off between completeness and computational efficiency [17, 16, 23]. This surge in innovation has also led to the establishment of competitions such as VNNComp [4], which systematically evaluate the performance of neural network verification tools. Notable examples include NNenum [1], Marabou [17, 31], and PyRAT [9], as well as frameworks such as MnBAB [10] (which builds upon ERAN [23] and PRIMA [21]) and α, β-CROWN [27, 32], the winner of the last 4 VNNComp, based on branch-and-bound methodology [34, 5].

However, these verification tools focus on *local* robustness: given a DNN, an image, and a small neighborhood around this image, they verify whether all images in the neighborhood are assigned the same classification. This neighborhood

is provided by a maximal perturbation of the input image, often an L_∞-norm constraint. Under this metric, every subpixel of the input image can vary in a very small range, typically $\frac{2}{255}$ (that is, 2 levels of gray/blue/red/green). Although it is not necessarily the most meaningful perturbation, L_∞ is the usual choice because it is perfectly linear (subpixel perturbations are independent), which is easier to verify.

Crucially, however, these verification tools for local robustness are too computationally intensive for real-time decision-making pipelines. For instance, consider an autonomous vehicle processing a live video feed: images of the feed cannot be certified as robust in a few ms on embedded hardware. This prevents the development of safety filters that could skip non-robust images and only consider certified robust images.

In this paper, we consider the verification of *global* robustness, that is, we do not restrict the verification process to the neighborhood of a specific, fixed input. Our approach to achieve real-time verification employs a two-step procedure:

Global bound analysis. The first step, performed offline, computes global bounds on the maximum possible shift between output values of different decision classes due to a perturbation, following an approach similar to VHAGaR [14]. Specifically, for any two decision classes C and D and perturbation ε, we compute an upper bound $\bar{\beta}^{\varepsilon}_{C,D}$ on $\max_{I,I',|I-I'|\leq\epsilon}(x_C - x'_C + x'_D - x_D)$, where $x_{C,D}, x'_{C,D}$ are the output values of class C, D from image I and I'. Unlike local robustness methods, which require k separate, computationally-expensive calls to certify k images, the global bound $\bar{\beta}^{\varepsilon}_{C,D}$ is computed offline once per network and remains valid across the entire input space.

Real-Time Verification. The second step occurs in real-time, after the DNN's inference phase. For a given input image I, we identify the class C with the highest output value x_C, and check whether for every other class $D \neq C$, $x_C - x_D > \bar{\beta}^{\varepsilon}_{C,D}$. If this is the case, then we can certify that image I is robust for perturbation ε, because an ε-perturbed image I' could at most get $x'_D \leq \bar{\beta}^{\varepsilon}_{C,D} - (x_C - x_D) + x'_C < x'_C$, hence C is also the predicted class for image I'. This check typically requires only a few dozen CPU instructions, which can be completed under $1ms$ on a single embedded CPU core. When an image fails this verification, one could either skip it (e.g., in a video stream) or revert to a safer, degraded mode until a trustworthy, robust image is received.

Our main contributions address the challenges of computing the *global bounds* $\bar{\beta}^{\varepsilon}_{C,D}$ for all pairs of classes (C, D):

1. We develop a novel *Diff* MILP encoding for the global robustness problem, where the variables are the values of the perturbed neurons, as well as the differences between the original and the perturbed neuron values (called the *diff variables*, introduced in [3]). We study and encode how the *diff variables* evolve after passing through a ReLU (Prop. 2), see Section 3.4. By comparison, the *classical* MILP model [26] employed in [14, 28, 29] computes the small values of the *diff variables* as the difference between 2 larger values, which is *asymptotically* exact but results in poor *practical* accuracy [3]: the *Diff* model is more accurate in practice, with the same number of variables.

2. Further, from the *Diff MILP model*, which is asymptotically exact, we develop two abstract MILP models, which are more efficient but also asymptotically less accurate than *Diff*. Namely, the *2b+1* model, accurate on the *diff variables* but abstract on the perturbed variables; while the *1b* model has a unique binary variable per ReLU, only considering the *diff variables*.
3. In terms of perturbations, we consider conjunctions of L_∞- and L_1-norms, which allow to accurately describe perturbations. For instance, "each subpixel is perturbed by at most $\frac{50}{255}$ (L_∞) and the sum of the absolute value of perturbations over all subpixels is at most 1" (L_1-perturbation). While L_1 perturbations are not linear (because of the absolute values), and thus they are seldom used, we show in Section 3.1 how to use them as a perturbation in the MILP model without incurring any expensive binary variables (only cheap linear variables are necessary).
4. Bounds obtained on the full input space are particularly pessimistic, as all inputs, including *Out of Distribution (OOD [15])* inputs far away from the training dataset, need to be accounted for. To address this, we consider principal component analysis (PCA) from engineering science [7]. Specifically, we use PCA to faithfully represent common inputs from the dataset. OOD inputs may be represented unfaithfully, which is reasonable as the DNN is unlikely to provide a reasonable answer on OOD inputs anyway. For instance, 20 linear components represent MNIST inputs faithfully: the PCA-DNN does not lose accuracy on the MNIST dataset.
5. Experimentally, the *Diff MILP model* computes upper bounds $\bar{\beta}^{\varepsilon}_{C,D}$ which reduces the gap to the lower bound compared with the optimized version of the classical MILP encoding (see Table 1); namely reducing the number of binary variables (VHAGaR [14]); or adding linear constraints from the *diff variables* (ITNE [29]). The abstractions *2b+1* and *1b* variants offer different trade-offs, reaching better bounds than the *Diff MILP* model when the instance is very complex, or when runtime is limited (Table 3). Further, using PCA reduces the upper bound $\bar{\beta}^{\varepsilon}_{C,D}$ by 3 to 20 times. Overall, using these different techniques (*Diff* MILP model, abstraction, and PCA) enables the real-time certification of $\geq 70\%$ of fresh images for an L_1-perturbation of $1, 4, 2$ over the MNIST, Fashion-MNIST, and CIFAR-10 datasets, respectively, see Table 4. The online process only adds 0.5 ms of latency per image, and 2000 images/second can be treated per CPU core.

1.1 Related Work on Global Robustness

While [3] does not tackle global robustness per se, it is one of the first works to consider *differential* verification, introducing the *diff* variables that are used here and also in [29]. The paper [3] did not consider MILP encoding. Some works [18, 33, 24, 6, 11] focus on training a network to be, rather than proving the DNN is, globally robust. Other works [2, 22, 12] estimate global robustness bounds, but without hard verification certificates, e.g., using probabilistic guarantees [19, 20].

More related, several works [17, 28–30] consider certifying *restricted* Lipschitz bound, to understand how the output can change given an input perturbation.

However, they do not consider whether the classification changes. Closest to our work is VHAGaR [14], which computes bounds $\bar{\alpha}^{\varepsilon}_{i,j}$ similar to our bounds $\bar{\beta}^{\varepsilon}_{i,j}$, but limited to L_∞-perturbations and variants. Crucially, VHAGaR bounds are not small enough for real-time robustness verification, whereas our bounds are.

Technically, none of the MILP-based works [14, 28, 29] consider L_1-perturbations or PCA. Further, they use the classical MILP encoding from [26], and not our novel *Diff* model, nor even the "1b" model. Note that [29] considers the *diff variables* $y, \hat{y}$, adding explicitly one *linear* constraint $\frac{y_i-\gamma_i}{2} \leq \hat{y}_i \leq \frac{y_i+\gamma_i}{2}$ implied by each of our three MILP models *Diff*, "2b+1" and "1b" models; Eq. (3) in [29] corresponding essentially to it. [14] does things differently, adding per neuron constraints (depending on the perturbation) between the (binary) variables to simplify the MILP problem, which is well-adapted to occlusion and patches but less to L_1-perturbations.

2 Notations and Preliminaries

In this paper, we will use lower case Latin a for scalars, bold $\boldsymbol{z}$ for vectors, capitalized bold $\boldsymbol{W}$ for matrices, similar to notations in [27]. To simplify the notations, we restrict the presentation to feed-forward, fully connected ReLU Deep Neural Networks (DNN for short), where the activation function is ReLU : $\mathbb{R} \to \mathbb{R}$ with $\mathrm{ReLU}(x) = x$ for $x \geq 0$ and $\mathrm{ReLU}(x) = 0$ for $x \leq 0$, which we extend componentwise on vectors.

An ℓ-layer DNN is provided by ℓ weight matrices $\boldsymbol{W}^i \in \mathbb{R}^{d_i \times d_{i-1}}$ and ℓ bias vectors $\boldsymbol{b}^i \in \mathbb{R}^{d_i}$, for $i = 1, \ldots, \ell$. We call d_i the number of neurons of hidden layer $i \in \{1, \ldots, \ell - 1\}$, d_0 the input dimension, and d_ℓ the output dimension.

Given an input vector $\boldsymbol{z}^0 \in \mathbb{R}^{d_0}$, denoting $\hat{\boldsymbol{z}}^0 = \boldsymbol{z}^0$, we define inductively the value vectors $\boldsymbol{z}^i, \hat{\boldsymbol{z}}^i$ at layer $1 \leq i \leq \ell$ with

$$\boldsymbol{z}^i = \boldsymbol{W}^i \cdot \hat{\boldsymbol{z}}^{i-1} + \boldsymbol{b}^i \qquad\qquad \hat{\boldsymbol{z}}^i = \mathrm{ReLU}(\boldsymbol{z}^i).$$

The vector $\hat{\boldsymbol{z}}$ is called post-activation values, $\boldsymbol{z}$ is called pre-activation values, and $\boldsymbol{z}^i_j$ is used to denote the j-th neuron in the i-th layer. For $\boldsymbol{x} = \boldsymbol{z}^0$ the (vector of) input, we denote by $f(\boldsymbol{x}) = \boldsymbol{z}^\ell$ the output.

In this paper, we consider the *global* verification problem, where we optimize over all images I and perturbations I' of I with $|I - I'| \leq \varepsilon$. We will consider three kinds of variables:

- $x_j, \hat{x}_j$, for neurons j with input image I,
- $x'_j, \hat{x}'_j$, for neurons j with input the perturbed I',
- $y_j = x_j - x'_j$ the *diff variable*, with $\hat{y}_j = \hat{x}_j - \hat{x}'_j$ (and *not* $\hat{y}_j = \mathrm{ReLU}(x_j - x'_j)$), as first proposed by [3].

2.1 MILP encoding for local ReLU

Mixed Integer Linear Programming (MILP) can encode ReLU DNNs faithfully: For an *unstable* neuron n, that is, with values $x \in [\mathrm{LB}(n), \mathrm{UB}(n)]$ with $\mathrm{LB}(n) <$

$0 < \mathrm{UB}(n)$, the value $\hat{x}$ of $\mathrm{ReLU}(x)$ can be encoded exactly in an MILP formula with one binary/integer variable a valued in $\{0, 1\}$, using constants $\mathrm{UB}(n), \mathrm{LB}(n)$ with 4 constraints [26]:

$$\hat{x} \geq x \wedge \hat{x} \geq 0 \wedge \hat{x} \leq \mathrm{UB}(n)a \wedge \hat{x} \leq x - \mathrm{LB}(n)(1-a) \tag{1}$$

Proposition 1. *[26] A solution $x, \hat{x}, a$ of the above MILP program (1) satisfies $\hat{x} = \mathrm{ReLU}(x)$, and $a = 1$ if $x > 0$ and $a = 0$ if $x < 0$ (both $a = 0, 1$ are possible if $x = 0$).*

The encoding from $(\hat{x}_j)_{j \text{ in layer } i}$ to $(x_{j'})_{j' \text{ in layer } i+1}$ variables is simply the linear combination $\boldsymbol{z}^i = \boldsymbol{W}^i \cdot \hat{\boldsymbol{z}}^{i-1} + \boldsymbol{b}^i$. Hence, the full MILP model uses exactly as many binary variables as unstable ReLUs.

2.2 LP relaxation

MILP instances can be linearly relaxed into LP over-abstraction, where variables a originally restricted to integers in $\{0, 1\}$ (binary) are relaxed to real numbers in the interval $[0, 1]$, while maintaining the same encoding. Since solving LP instances takes polynomial time, this optimization is significantly more efficient. However, this efficiency comes at the cost of precision, often resulting in less stringent bounds. This approach is termed the *LP relaxation*.

Intermediate between these 2 extreme cases, there is *partial MILP* (pMILP for short) to get trade-offs between accuracy and runtime [13]: Let X be a subset of the set of unstable neurons, and n a neuron for which we want to compute upper and lower bounds on values: the pMILP based on X to compute neuron n uses the MILP encoding (1), where variable a is:

- binary for neurons in X (exact encoding of the ReLU),
- linear for neurons not in X (linear relaxation).

3 MILP Models for Global Robustness

Let N be the number of neurons of a DNN. Compared with local robustness which considers only variables $(x'_j)_{j \leq N}$ for the perturbed image I', global robustness considers also variables $(x_j)_{j \leq N}$ for the (*non*-fixed) input image I, and thus necessitates double the number of (binary) variables (for I and for I'). Worst-case complexity being exponential in the number of binary variables, global robustness (e.g., ITNE and VHAGaR [29, 14]) is much harder than local robustness.

3.1 Encoding L_1-perturbations for input variables

The standard encoding of L_∞-perturbations bounded by ε uses linear constraint $x_j - \varepsilon \leq x'_j \leq x_j + \varepsilon$ for each *input* neuron j.

We now explain how to encode L_1-perturbations $\sum_{j\leq n} |x_j - x'_j| \leq \varepsilon$, using additional variables that are linear only. Let n be the number of input neurons. We use n additional linear variables $(A_j)_{j\leq n}$, with the following constraints:

$$\begin{cases} A_j \geq x_j - x'_j & \text{for all } j \leq n \\ A_j \geq x'_j - x_j & \text{for all } j \leq n \\ \sum_{j\leq n} A_j \leq \varepsilon \end{cases} \tag{2}$$

Proof. The first two lines of constraints (2) are equivalent with $A_j \geq |x_j - x'_j|$. So the whole constraint (2) implies that $\sum_{j\leq n} |x_j - x'_j| \leq \varepsilon$.

Conversely, for every instance that satisfies $\sum_{j\leq n} |x_j - x'_j| \leq \varepsilon$, then there is also an instance which satisfies constraints (2), by defining $A_j = |x_j - x'_j|$ for all $j \leq n$. This shows that (2) is equivalent with $||\boldsymbol{x} - \boldsymbol{x}'||_{L_1} = \sum_{j\leq n} |x_j - x'_j| \leq \varepsilon$.

3.2 Constraints for propagation in the DNN

There are two kinds of propagation of values for variables: for a neuron j of layer ℓ, the value of variable x_j is defined by the constraint $x_j = \sum_{i\in\ell-1} \alpha_i \hat{x}_i$, which is a simple linear operation from the output $(\hat{x}_i)_{i\in\ell-1}$ of neurons i in layer $\ell - 1$.

In the same way, $x'_j = \sum_{i\in\ell-1} \alpha_i \hat{x}'_i$ for variable x'_j associated with the perturbed image I', using the same α_i associated with the DNN weights. Similarly, if the *Diff* variables $y_j = x_j - x'_j$ are considered as defined in Section 2, with $\hat{y}_j = \hat{x}_j - \hat{x}'_j$, then we simply have $y_j = \sum_{i\in\ell-1} \alpha_i \hat{y}_i$.

It remains to consider the (non-linear) ReLU functions from x_j, x'_j and/or y_j to $\hat{x}_j, \hat{x}'_j$ and/or $\hat{y}_j$, which is the complex part, with different possible solutions explored in the following with different sets of variables considered.

3.3 The classical encoding of ReLUs for global robustness

The classical encoding is used (with variants) by VHAGaR [14] and ITNE [29]. For each neuron j, variables $x_j, \hat{x}_j$ and $x'_j, \hat{x}'_j$ are considered. For each ReLU j, a binary variable a_j and related constraints from the classical encoding (1) [26] are used to set $\hat{x}_j$, and an *independent* binary variable a'_j and related constraints from the classical encoding are used to set $\hat{x}_j$. We call this encoding $\mathcal{M}^{classical}$.

The key values to compute are the *diff*erential values of y_j between x_j and x'_j. Most of the values of y_j are likely small because they are inherited from a small perturbation between I and I'. However, $\mathcal{M}^{classical}$ only maintains x_j and x'_j separately, and both can take large values. This results into poor evaluation of y_j. This problem was first raised and experimentally demonstrated in [3].

Similarly, the standard way to make the MILP model more efficient is to linearly relax some of the binary variables (see Section 2). The problem is that the LP relaxation of $\mathcal{M}^{classical}$ is extremely inaccurate, as variables x_j, x'_j dependencies are due solely on the function definition through common ancestor variables in previous layers - as this is approximated with LP, most of the dependencies will be lost. On top of being an issue with partial MILP models, LP

relaxation is also used during the Branch and Bound process internal to MILP solvers. So even without explicit LP relaxation, the classical MILP encoding of [26], while asymptotically accurate, will result in looser bounds in practice.

To improve this, ITNE adds some linear constraints to the model (Eq. (3) in [29], see also Section 3.4), while VHAGaR reduces the number of binary variables depending on the perturbation used [14].

3.4 The novel *Diff* MILP encoding for global robustness

We now present our main methodological contribution, the *Diff* MILP encoding. It considers variables $x'_j, \hat{x}'_j, y_j, \hat{y}_j$, where $y_j, \hat{y}_j$ are the *Diff variables* [3], with $y_j = x_j - x'_j$ and $\hat{y}_j = \mathrm{ReLU}(x_j) - \mathrm{ReLU}(x'_j)$. A first important observation is that, since the image I and the deformation I' have perfectly symmetrical roles and assuming that γ_j is an *upper* bound of y_j, then $-\gamma_j$ is a *lower* bound for y_j.

To encode the equation describing $\hat{x}'_j$, we use the MILP constraints from the classical encoding (1), including one binary variable a'_j, similar to ITNE.

However, unlike ITNE, the *Diff* MILP model encodes the equation for $\hat{y}_j$ directly, without resorting to the classical encoding from $x_j (= y_j + x'_j)$ to $\mathrm{ReLU}(x_j)$ and using $\hat{y}_j = \mathrm{ReLU}(x_j) - \mathrm{ReLU}(x'_j)$, which is correct but does not encode explicitly that $\hat{y}_j$ is small when y_j is small, since both ReLUs can be large and they are encoded independently, see also [3]. Instead, we study the function from (x'_j, y_j) to $\hat{y}_j$, and encode it into the novel MILP *Diff* model.

We first display on Figure 1 the function $f(x'_j, y_j) = \hat{y}_j$ for $y_j \in [-\gamma_j, \gamma_j]$. When $x'_j \leq 0$ and $x_j = x'_j + y_j \leq 0$, we have $\hat{y}_j = \mathrm{ReLU}(x_j) - \mathrm{ReLU}(x'_j) = 0 - 0 = 0$. For $x'_j \geq 0$ and $x_j = x'_j + y_j \geq 0$, we have $\hat{y}_j = \mathrm{ReLU}(x_j) - \mathrm{ReLU}(x'_j) = x_j - x'_j = y_j$. Between these two linear planes, we have two more linear planes presenting intermediate cases, so 4 linear planes in total.

Recall first that a'_j is the binary variable used in the classical MILP encoding for $\hat{x}'_j = \mathrm{ReLU}(x'_j)$. We set $a' = a'_j$ (whose value is already settled, see Proposition 1), and use it in the MILP encoding of $\hat{y}_j = \mathrm{ReLU}(x_j) - \mathrm{ReLU}(x'_j)$:

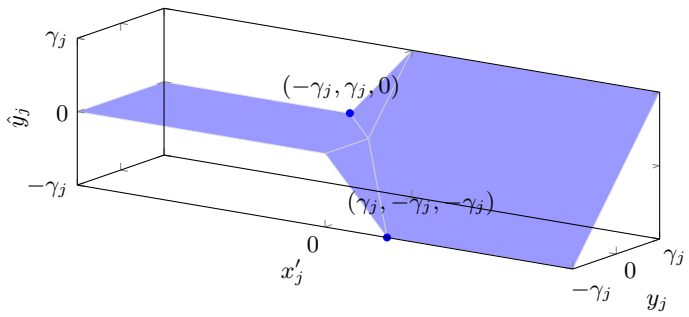

Fig. 1: Function $(y_j, x'_j) \mapsto \hat{y}_j = \mathrm{ReLU}(x_j = y_j + x'_j) - \mathrm{ReLU}(x'_j)$

Proposition 2. *Assuming $y_j \in [-\gamma_j, \gamma_j]$ and $x'_j, y_j + x'_j \in [\alpha_j, \beta_j]$, we have that $\hat{y}_j = \mathrm{ReLU}(x_j = y_j + x'_j) - \mathrm{ReLU}(x'_j)$ is the solution of the MILP program:*

$$
\begin{array}{ll}
y_j + x'_j \leq a\beta_j & y_j + x'_j \geq (1-a)\alpha_j \\
x'_j \leq a'\beta_j & x'_j \geq (1-a')\alpha_j \\
\hat{y}_j \leq a\gamma_j & \hat{y}_j \geq -a'\gamma_j \\
\hat{y}_j \leq y_j + x'_j - (1-a)\alpha_j & \hat{y}_j \geq y_j + x'_j - a'\beta_j \\
\hat{y}_j \leq -x'_j + a\beta_j & \hat{y}_j \geq -x'_j + (1-a')\alpha_j \\
\hat{y}_j \leq y_j + (1-a)\gamma_j & \hat{y}_j \geq y_j - (1-a')\gamma_j
\end{array}
$$

where $a, a' \in \{0,1\}$ are binary variables, and a' is shared with the classical MILP encoding for $\hat{x}'_j = \mathrm{ReLU}(x'_j)$.

So to encode both $\hat{y}_j$ and $\hat{x}'_j$, the *Diff* encoding needs 2 binary variables, similar to the classical encoding for $\hat{x}_j$ and $\hat{x}'_j$.

Proof (sketch). First, we know the constraints for $\hat{x}'_j = \mathrm{ReLU}(x'_j)$ are exact, by Prop 1. We now do a case analysis depending on the value of both binary variables. We check that in all 4 cases $\hat{y}_j = \mathrm{ReLU}(y_j + x'_j) - \mathrm{ReLU}(x'_j)$.

$\mathbf{a = 0, a' = 0}$**:** In this case $y_j + x'_j \leq 0$ and $x'_j \leq 0$ (lines 1&2). We thus have to show that $\hat{y}_j = 0$. This is true thanks to inequalities in line 3.

$\mathbf{a = 1, a' = 0}$**:** In this case $y_j + x'_j \geq 0$ and $x'_j \leq 0$ (lines 1&2). We thus have to show that $\hat{y}_j = y_j + x'_j$. This is true thanks to inequalities in line 4.

$\mathbf{a = 0, a' = 1}$**:** In this case $y_j + x'_j \leq 0$ and $x'_j \geq 0$ (lines 1&2). We thus have to show that $\hat{y}_j = -x'_j$. This is true thanks to inequalities in line 5.

$\mathbf{a = 1, a' = 1}$**:** In this case, $y_j + x'_j \geq 0$ and $x'_j \geq 0$ (lines 1&2). We thus have to show that $\hat{y}_j = y_j$. This is true thanks to inequalities in line 6.

Equations not mentioned are moot, as shown in the appendix. □

Intuitively, the *diff values* will be small as they will be carrying the difference between the image I and its slight perturbation I'. The LP relaxation of $\mathcal{M}^{classical}$ will miss this totally, while the LP relaxation of the Diff model contains such important bounds, including (see section 3.6 and Fig. 2):

$$\frac{y_j - \gamma_j}{2} \leq \hat{y}_j \leq \frac{y_j + \gamma_j}{2} \tag{3}$$

This is a simplification (due to the noticed symmetry) of the additional linear constraint (3) of [29] used explicitly in ITNE.

3.5 A decoupled *2b+1* model

We propose a variant of the *Diff* encoding: the binary variables a' appearing in Prop. 2 can be decoupled from the binary variable a'_j appearing in the classical encoding of $\hat{x}'_j = \mathrm{ReLU}(x'_j)$. This would mean having 3 binary variables, which would be too costly. The variable a'_j used for x'_j is thus linearly relaxed, while

both variables a and $, a'$ used in the encoding of $\hat{y}_j$ are kept binary. We call this the *2b+1* model, which uses 2 binary variables per neuron, as the classical and *Diff* encodings. The decoupling between a' and a_j removes constraints in the MILP encoding, which helps the solver. This, however, also means that the *2b+1* model is not as accurate as the *Diff* model asymptotically.

3.6 The efficient *1b* model

To simplify further and limit the number of (binary) variables, we introduce an efficient *1b* model which only uses the diff variables $y_j, \hat{y}_j$. Again, we study the range of values $\hat{y}_j$ can take according to the value of y_j, as depicted in Fig. 2. This is a projection of Fig. 1 onto $y_j, \hat{y}_j$.

Proposition 3. *Given $y_j \in [-\gamma_j, \gamma_j]$, $\hat{y}_j$ is a solution of the MILP program :*

$$\hat{y}_j \leq a\gamma_j \qquad \hat{y}_j \geq y_j - a\gamma_j$$
$$\hat{y}_j \geq (a-1)\gamma_j \qquad \hat{y}_j \leq y_j + (1-a)\gamma_j$$

where $a \in \{0, 1\}$ is a binary variable.

We can easily check that the LP relaxation of this MILP program, depicted in broken lines on Fig. 2, is exactly (3). As the *1b* model is a projection of the *Diff* model and also of the *2b+1* models, the LP relaxations of the *Diff* and the *2b+1* models contain (3). Proof of Proposition 3 is in the appendix.

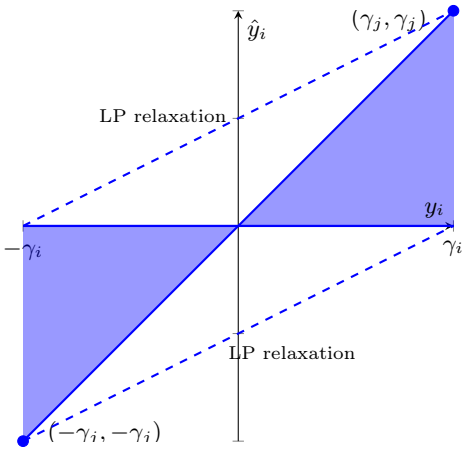

Fig. 2: Possible values of $\hat{y}_j$ depending on y_j for the *1b* model.

4 Real-time verification

As explained in the introduction, our objective is to compute upper and lower bounds for $\beta^{\varepsilon}_{C,D} = \max_{I,I',|I-I'|\leq\epsilon}(x_C - x'_C + x'_D - x_D) = \max_{|I-I'|\leq\varepsilon}(y_C - y_D)$, where $x_{C,D}, x'_{C,D}$ are the output values of class C, D from image I to the perturbed image I' or the differential between I, I' output values, respectively.

Upper bounds $\bar{\beta}^{\varepsilon}_{C,D} \geq \beta^{\varepsilon}_{C,D}$ will allow us to verify in real-time whether a new incoming image can be certified to be robust for the perturbation. Lower bounds $\underline{\beta}^{\varepsilon}_{C,D} \leq \beta^{\varepsilon}_{C,D}$ allow us to evaluate how large the gap $(\bar{\beta}^{\varepsilon}_{C,D} - \underline{\beta}^{\varepsilon}_{C,D})/\underline{\beta}^{\varepsilon}_{C,D}$ is. MILP solvers, e.g., Gurobi, provide both an upper bound ("bound") $\bar{\beta}^{\varepsilon}_{C,D}$ as well as a lower bound ("solution") $\underline{\beta}^{\varepsilon}_{C,D}$.

Related bounds were proposed in [14], for an evaluation purpose only (how narrow is the gap) rather than to verify robustness in real-time. Namely, VHAGaR [14] defines $\alpha^{\varepsilon}_{C,D}$, computing the perturbation necessary to switch an image classified as class C into an image classified as class D:

$$\alpha^{\varepsilon}_{C,D} = \max_{|I-I'|\leq\varepsilon}(x_C - \max_{E\neq C} x_E) \qquad \text{s.t. } x'_D \geq \max_{E\neq D} x'_E \wedge x_C \geq \max_{E\neq C} x_E$$

Notice that $\alpha^{\varepsilon}_{C,D}$ is not symmetrical, while $\beta^{\varepsilon}_{C,D}$ is.

Assume we have computed (offline) $\bar{\beta}^{\varepsilon}_{C,D}$ for each C, D. Now in real-time:

0 Run inference on the DNN to obtain the values x_D of output neuron for each class D. Let C be the class with maximal x_C, that is, the class predicted by the DNN. These are the basic inference operations standardly performed for classification by a DNN.
1 Compute for each class $D \neq C$ a bit $b_D = 1$ iff $x_D + \bar{\beta}^{\varepsilon}_{C,D} > x_C$.
2 Return "certified robust" iff $\bigvee_{D\neq C}(b_D) = 0$.

Proposition 4. *If "certified robust" is returned over input I, then for all perturbed input I' with $|I - I'| \leq \varepsilon$, the DNN has the same decision on I and on I', i.e. the DNN is robust around I for perturbation ε.*

Proof. Let C be the decision of the DNN on I. Assume by contradiction that $D \neq C$ is the decision on I'. We know that $x_D + \bar{\beta}^{\varepsilon}_{C,D} < x_C$ because "certified robust" is returned over input I. That is, $x_D - x_C + \bar{\beta}^{\varepsilon}_{C,D} < 0$

By definition of $\bar{\beta}^{\varepsilon}_{C,D}$, we have $x'_D - x'_C + x_C - x_D \leq \bar{\beta}^{\varepsilon}_{C,D}$. ($I$ and I' have symmetrical roles), that is $x'_D - x'_C \leq x_D - x_C + \bar{\beta}^{\varepsilon}_{C,D} < 0$. That is $x'_D < x'_C$, a contradiction with D is the decision on I'. □

The overhead (steps 1,2 above) on top of the standard inference step 0, amounts to $2k - 1$ operations, where k is the number of classes. That is, 19 operations in total for $k = 10$ classes (cases of MNIST, Fashion MNSIT and CIFAR-10). More precisely: $(k-1)$ ADD($+$), $(k-1)$ COMP($<$) and 1 OR($\bigvee$) of $(k-1)$ bits. This can be performed extremely fast, even on limited hardware.

5 Principal Component Analysis for Bound Computation

Global upper bounds $\bar{\beta}$ obtained across the entire input space tend to be overly pessimistic, since they must accommodate all possible inputs, including out-of-distribution (OOD) [15] samples far from the training dataset. Worst cases $\underline{\beta}$ for β are indeed produced by such OOD samples, see Fig. 3. To address this, we employ principal component analysis (PCA) [7], a technique from engineering science, to restrict the output (hence lowering the bounds) *without* restricting the input. Specifically, PCA extracts the most important linear components of a dataset by computing linear operators to project data from the full input dimension to a reduced feature space (PCA-encoding) and back (PCA-decoding).

We construct the *PCA-DNN* by prepending PCA-encoding and decoding to the original DNN, transforming an image into its *PCA-representation*, see Fig. 5. The PCA-representation is faithful for common *in-distribution (ID)* inputs (e.g., Fig. 4), though it may become unfaithful for OOD inputs. Since DNNs are inherently unreliable on OOD inputs, misrepresenting them does not impact accuracy, which we check empirically. Crucially, the PCA-DNN inputs are not restricted: results from the PCA-DNN real-time verification hold for any arbitrary image, even OOD. The PCA-pipeline is as follows (see Fig. 5 for an example on MNIST):

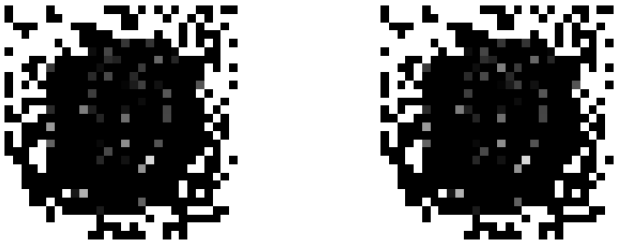

Fig. 3: An (OOD) image for MNIST and its perturbation (in a unique pixel at x=14, y=8 from top) maximizing $\underline{\beta}^{.5}_{-6,8} = .518$, as obtained by the *Diff* model.

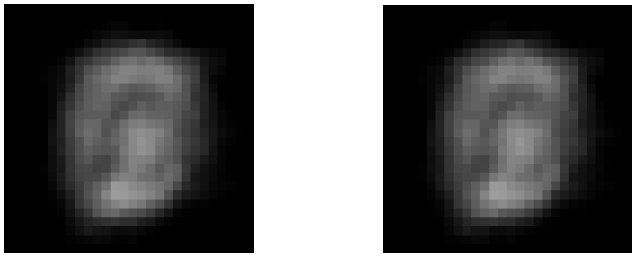

Fig. 4: An (ID) image for MNIST and its perturbation when using the PCA-representation, maximizing $\underline{\beta}^{.5}_{-6,8} = .084$, as obtained by the*Diff* model.

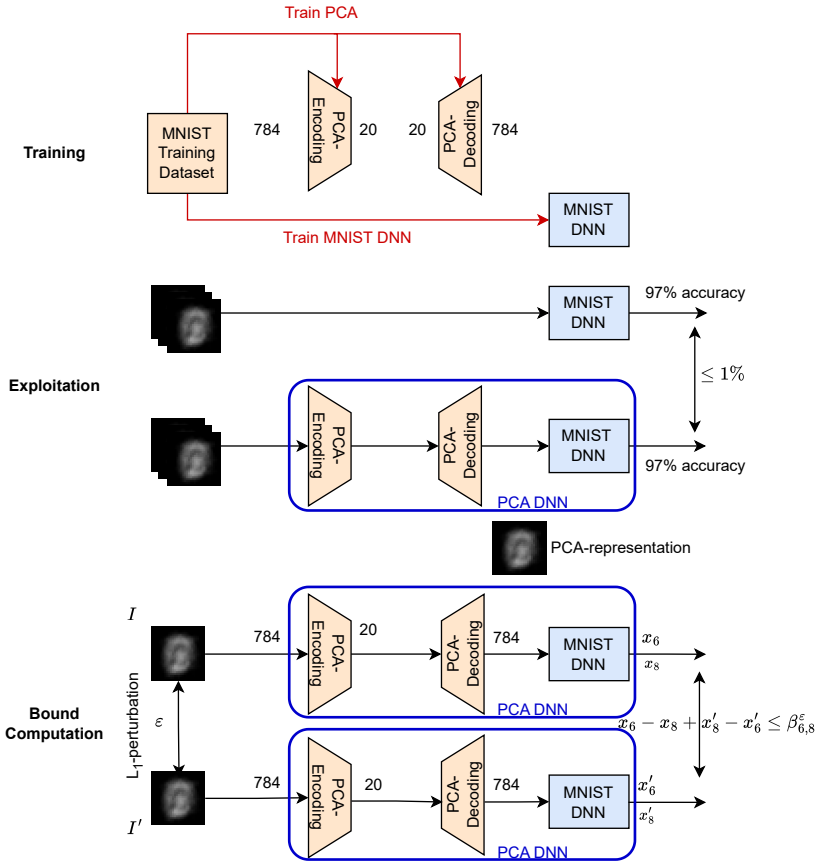

Fig. 5: The pipeline training and exploiting the **PCA-DNN** on MNIST dataset.

1. *Training* the PCA-encoding and decoding on the training dataset (e.g., MNIST, FMNIST, CIFAR10). Note that the original DNN has been previously learned on this training dataset.
2. *In Exploitation*, the **PCA-DNN** is formed by calls to PCA-encoding, then PCA-decoding (forming the PCA-representation of the input), and then the original DNN. Although PCA-representations are full size (784 pixels for MNIST), they lie in a reduced space. We check the accuracy of the PCA-DNN, that is, the % of images where the predicted class matches the ground truth. We set the PCA dimension as small as possible (for efficiency) while ensuring that the PCA-DNN accuracy loss is minimal ($\leq 1\%$) wrt the original DNN without PCA (20 for MNIST in Fig. 5).
3. *In Bound Computation*, we treat the pixels of the original image I and the perturbed image I' as variables, along with the reduced PCA dimensions

(20×2 from I and I'). These are then fed to the first layer of the DNN (linear transformation). Compared to computing bounds on the original DNN, we only add *linear* variables, as few as 2x the reduced PCA dimensions. The associated constraints, which come from the PCA-encode/decode, are also linear. Next, we have the exact same MILP encoding (e.g., *Diff* encoding) for the DNN. Hence, the number of *binary* variables for PCA-DNN is the same as without PCA. Finally, the goal is to optimize, e.g., $\beta^{\varepsilon}_{6,8}$, the value of $y_6 - y_8 = x_6 - x'_6 + x'_8 - x_8$, where y_C is the *diff* variable of the output neuron for class $C \in \{6, 8\}$.

We report in Fig. 3 the image and perturbed image reaching the maximal lower bound $\underline{\beta}^{.5}_{6,8} = 0.518$ obtained when using the original DNN (without PCA). They are OOD inputs to the MNIST digit classification dataset. In contrast, there are natural (ID) images and perturbed images reaching the maximal lower bound $\underline{\beta}^{.5}_{6,8} = 0.084$ of the *PCA-DNN* (Fig. 4): among all the images producing a given worst-case PCA-representation, we can choose the PCA-representation itself, which is ID. Note that the lower bound $\underline{\beta}$ is lower for the PCA-DNN, which is perfectly normal as we restrict the network's *output* to a reduced space close to the MNIST dataset. This will also be the case for the upper bound $\overline{\beta}$, which is very desirable as it improves robustness certification.

6 Experimental Evaluation

We implemented our code in Python 3.8. Gurobi 9.52 was used for solving the MILP problems. We conducted our evaluation on an AMD Threadripper 7970X (32 cores@4.0GHz) with 256 GB of main memory and 2 NVIDIA RTX 4090 (not used, Gurobi is CPU only). ITNE [29] was reimplemented using the same path as our code, except for the MILP variable x_j using the classical MILP encoding plus the extra linear constraint Eq. (3). The Julia VHAGaR code from `https://github.com/ananmkabaha/VHAGaR` [14] was run on the same hardware.

We consider 4 pretrained DNNs on 3 different datasets (MNIST, Fashion-MNIST, and CIFAR10). Namely, MNIST-FC, a fully connected DNN available at `https://github.com/eth-sri/eran` ("5x100"). We also considered 3 convolutional DNNs: MNIST-Conv, FMNIST-Conv, and CIFAR10-Conv, from [14] ("conv1"). We will also consider 3 PCA-DNNs, as detailed in Section 6.2.

6.1 Comparison for global robustness with VHAGaR and ITNE (L_∞)

We use VHAGaR benchmarks to be able to compare: VHAGaR bound α (see Section 4), perturbation $L_\infty = 0.05$, VHAGaR convolutional DNNs, for which the VHAGaR reduction rules of binary variables are implemented. We tested with 1000s and 14400s (4h) timeout. We report the upper bound $\bar{\alpha}$ each MILP model finds. For each DNN, we report the best lower bound $\underline{\alpha}$ among the MILP models, and report the gap between $\bar{\alpha}$ and $\underline{\alpha}$ as % of increase over $\underline{\alpha}$. We report

$L_\infty = 0.05$	Timeout	MILP model:	VHAGaR	ITNE	*Diff*	*2b+1*	*Best*
MNIST-Conv LB $\underline{\alpha}$= 9.51	1000s	up. Bound $\overline{\alpha}\downarrow$	20.80	12.32	*11.14*	**10.95**	***10.85***
		(% of gap $\downarrow$)	(120%)	(29%)	(*17*%)	(**15**%)	(***14***%)
	14400s	up. Bound $\overline{\alpha}\downarrow$	17.54	11.15	***10.14***	*10.29*	***10.07***
		(% of gap $\downarrow$)	(84%)	(17%)	(***7***%)	(*8*%)	(***6***%)
FMNIST-Conv LB $\underline{\alpha}$= 22.21	1000s	up. Bound $\overline{\alpha}\downarrow$	41.52	26.61	***23.27***	*24.09*	***22.78***
		(% of gap $\downarrow$)	(93%)	(20%)	(***5***%)	(*10*%)	(***3***%)
	14400s	up. Bound $\overline{\alpha}\downarrow$	37.26	24.06	***22.42***	*23.80*	***22.34***
		(% of gap $\downarrow$)	(72%)	(9%)	(***1***%)	(*9*%)	(***0.6***%)
CIFAR-Conv LB $\underline{\alpha}$= 19.04	1000s	up. Bound $\overline{\alpha}\downarrow$	36.57*	27.15	*25.94*	***24.52***	***23.29***
		(% of gap $\downarrow$)	(55*%)	(28%)	(*24*%)	(***20***%)	(***15***%)
	14400s	up. Bound $\overline{\alpha}\downarrow$	31.53*	22.73	***21.77***	*21.81*	***20.38***
		(% of gap $\downarrow$)	(39*%)	(12%)	(***10***%)	(***10***%)	(***5***%)

Table 1: Upper bound $\bar{\alpha}$ and (gap to lower bound $\underline{\alpha}$), lower is better, for $L_\infty = 0.05$ perturbation, averaging bounds $\bar{\alpha}_{C,D}$ over all pair of classes (C, D). *Best* computes the average $\bar{\alpha}$ using the lowest $\bar{\alpha}_{C,D}$ computed by *Diff* and *2b+1* for each (C, D). Lower gaps using *Best* means that there are cases (C, D) where *Diff* is much better than *2b+1*, and cases where it is the opposite. * means that VHAGaR failed to compute bounds for some cases (C, D) of CIFAR-Conv. For such cases, we used numbers from ITNE instead (which is overperforming VHAGaR in every test), so these numbers are an optimistic evaluation for VHAGaR.

results in Table 1. Recall that most robustness tools, e.g., α, β-Crown, can only handle *local* robustness and cannot be tested.

Discussion: This test is relatively easy, where the best upper bound $\bar{\alpha}$ found is close to the lower bound $\underline{\alpha}$ found, with at most a 6% gap. Using the *1b* model is not useful, as its abstraction will lose more than 6%. VHAGaR computes upper bounds $\bar{\alpha}$ quite far from the lower bound, with a gap $40 - 120\%$. Although we are using its benchmarks, VHAGaR methodology is not well-suited for L_∞ (it is more efficient on *Patch* [14]): ITNE is better, with gaps from $9 - 17\%$. Our method provides the *Best* overall gaps: $.6 - 6\%$, significantly tighter. Breaking up numbers, *2b+1* is more accurate for shorter timeout, and *Diff* is more accurate when there is sufficient time, as expected. Anyway, both are better for particular classes (C, D), as *Best* is lower than both *Diff* and *2b+1* in every test.

6.2 Experimental results for $L_1 = 1$ and PCA-DNN.

We then turn to testing with L_1 perturbations and comparing between DNNs and their PCA-DNN versions (which are not supported in VHAGaR). Here, we consider MNIST-FC, which has a higher accuracy than MNIST-Conv, to test a different architecture than convolutional Networks.

We recall that PCA-DNNs are generated following the process described in Section 5: for each PCA dimension, we report in Table 2 the accuracy loss from using PCA-DNN rather than the original DNN. We set the dimension at the minimal dimension that results in at most 1% loss of accuracy, that is $20, 25$ and 60 for MNIST-FC, FMNIST-Conv and CIFAR10-Conv respectively.

PCA dimension:	15	20	25	30	35	40	...	60
PCA-MNIST-FC	2%	0%	0%	0%	0%	0%		0%
PCA-FMNIST-Conv	5%	3%	1%	0%	0%	0%		0%
PCA-CIFAR10-Conv	16%	14%	12%	10%	14%	5%		1%

Table 2: Loss of accuracy of PCA-DNN vs DNN (no PCA) according to the PCA dimension. Blue corresponds to the minimal dimension ensuring loss $\leq 1\%$.

We consider the $\bar{\beta}$ bounds reached for L_1-perturbation at most 1, on the 3 original DNNs MNIST-FC, FMNIST-Conv and CIFAR10-Conv, plus their PCA variants for the dimension computed in Table 2. We report the results in Table 3, in the same format as Table 1.

Discussion: Compared with L_∞ (Table 1), the gap between the upper bound $\bar{\beta}$ and the lower bound $\underline{\beta}$ is much larger with L_1 perturbations, from 1.5 to 600. Using the faster *1b* model now becomes pertinent. In half of the cases, *1b* actually produces the best results. When the timeout is large enough compared to the complexity of the model, 2*b*+1 and sometimes *Diff* provide better bounds. ITNE is significantly worse, producing 25% worse bounds $\bar{\beta}$ on average than *Best*.

Comparing with and without PCA, as expected, PCA-DNNs allow reaching much smaller upper bounds $\bar{\beta}$ than without PCA, 15x for MNIST-FC, 3x for FMNIST-Conv, and 7x for CIFAR10-Conv. MNIST-FC is harder to solve than FMNIST-Conv, although they are comparable benchmarks, because fully connected architectures are richer than convolutional architectures of the same size. PCA-CIFAR10-Conv is the hardest test, because the dimension (60) is not as reduced as for MNIST and FMNIST (20 and 25).

$L_1 = 1$	Timeout	ITNE	*Diff*	*2b+1*	*1b*	*Best*
MNIST-FC	1000s	45.6	38.88	38.68	**37.34**	**37.33**
LB $\underline{\beta}$= 0.50	14400s	45.3	37.24	36.96	**33.68**	**33.67**
PCA-MNIST-FC	1000s	2.52	**2.43**	**2.43**	2.68	**2.41**
LB $\underline{\beta}$= 0.12	14400s	2.33	2.18	**2.00**	2.46	**2.00**
FMNIST-Conv	1000s	12.27	8.90	**7.55**	9.84	**7.55**
LB $\underline{\beta} = 4.16$	14400s	11.03	7.43	**6.37**	9.62	**6.22**
PCA-FMNIST-Conv	1000s	3.64	3.02	3.02	**2.74**	**2.74**
LB $\underline{\beta} = 0.96$	14400s	2.67	2.34	**1.95**	2.52	**1.95**
CIFAR10-Conv	1000s	10.49	8.30	**8.14**	8.17	**7.77**
LB $\underline{\beta} = 3.38$	14400s	10.49	**7.19**	7.45	7.31	**6.67**
PCA-CIFAR10-Conv	1000s	1.70	1.63	1.54	**1.28**	**1.28**
LB $\underline{\beta}$= 0.0015	14400s	1.67	1.56	1.52	**1.15**	**1.15**
extra long timeout	216000s	1.66	1.53	1.36	**0.99**	**0.99**

Table 3: Comparison of upper bound $\bar{\beta}$ (lower is better $\downarrow$) averaged over all bounds $\bar{\beta}_{i,j}$, between ITNE and *Best* of *Diff, 2b+1 and 1b*, for original DNNs as well as PCA-DNNs over MNIST, FMNIST and CIFAR datasets.

Benchmark	$L_1 = 1$	$L_1 = 2$	$L_1 = 3$	$L_1 = 4$	Latency overhead
MNIST-FC	0%	0%	0%	0%	0.5 ms
PCA-MNIST-FC	80%	21%	1%	0%	0.5 ms
FMNIST-Conv	75%	50%	28%	12%	0.5 ms
PCA-FMNIST-Conv	92%	86%	79%	70%	0.5 ms
CIFAR10-Conv	39%	0%	0%	0%	0.5 ms
PCA-CIFAR10-Conv	92%	72%	61%	51%	0.5 ms

Table 4: Percentage of images certified robust in real-time using the Best computed $(\beta^1_{i,j})_{i<j\leq 10}$, for different values of L_1-perturbations. Blue depicts the largest perturbation ensuring $\geq 70\%$ of images being robust. The latency overhead is the same as the number of classes (10) for all datasets.

6.3 Real-Time verification of Robustness

We now turn to how many images can be certified robust in real-time using the *Best* bounds $\beta^1_{i,j}$ computed in the previous subsection. We report the% of images certified robust in Table 4, as well as the latency overhead to certify them: it is half a millisecond in every case (using a single CPU core), as it depends only on the number of outputs of the DNN, which is 10 for all 6 DNNs considered. It means 2000 images can be treated per second by each CPU core, which can be deemed *real-time*. We set a threshold of 70% to evaluate how large the L_1 perturbation can be while still certifying at least this threshold: a threshold much lower than 70% would mean labeling too many incoming images as uncertified, which would degrade the system too much.

With that, we can certify at least 70% of the images for all 3 PCA-DNN label for a L_1-perturbation of 1, and even up to perturbation 2 for CIFAR10 and even 4 for FMNIST. We are unable to do it for the non-PCA version, but in the case of FMNIST-Conv, with a perturbation 4 times smaller than for its PCA-version. Overall, PCA significantly improves real-time verification.

6.4 Experimental results for regression (Pipe strain)

Our *Diff* model is not only accurate for classification. It can also provide upper bounds on the output of DNN-regressor due to small perturbations of the input. We consider here a pipe system and its DNN regressor from [8], that predicts the plastic strain in 3000 points of a mesh over a pipe given the deformation of each of these 3000 points. Compared to the classification tasks where the DNN is learned in a standard way and PCA is later applied to reduce the output dimension without compromising the accuracy, the DNN regressor has been learned on the reduced PCA basis directly, for both the high dimensional input and the high dimensional output (3000 dimensions each).

Figure 6 describes the pipeline used to learn the DNN regressor for the pipe system. The DNN that has been learned has 2 fully connected layers, 50 neurons each, learned as a surrogate model, with 10 reduced input and 26 reduced output.

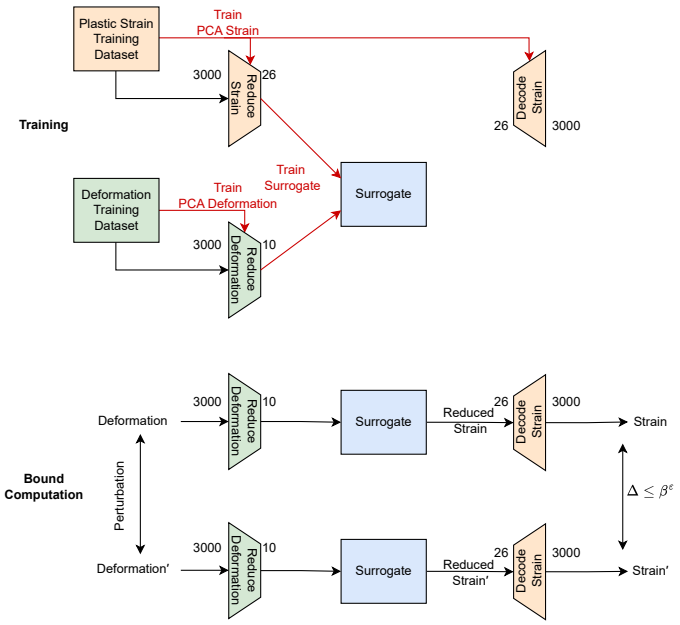

Fig. 6: Training and bound computation on the pipe use case. Two PCAs are used, for the input (deformation) and for the output space (plastic strain). The surrogate is learned from reduced deformation input to reduced strain output.

In terms of perturbation, a conjunction of $L_1 \leq 3.9$ and $L_\infty \leq .02$ perturbations, physically pertinent dimensions, and maximizes the sum of the difference in output of 10 selected points in the mesh between input and its deformation.

For the pipe system, we compare in Table 5 the different models ITNE, *Diff*, $2b+1$,$1b$ to produce bounds on the sum of the difference of strain over 10 specific points of the mesh, for a physically relevant perturbation of the deformation. The bounds we found are quite accurate, with a best bound of 0.0329 obtained by the $2b+1$ model, slightly better than the bound 0.0337 found within the same time by the *Diff* model, and better than the bound found 0.356 by the $1b$ model, although this bound has been found 70 times faster due to the simpler model (*1b* converges after 1000s). ITNE is significantly worse than even the $1b$ with 1000

Pipe	Timeout	ITNE	*Diff*	*2b+1*	*1b*
	1000s	0.045	0.042	0.041	**0.036**
Surrogate	14400s	0.042	0.036	**0.035**	0.036
LB = .025	72000s	0.042	0.034	**0.033**	0.036

Table 5: Comparison of upper bound (lower is better ↓) between *Diff*, *2b+1*, *1b* with ITNE on the pipe system with timeouts of 1000s, 14440s and 72000s.

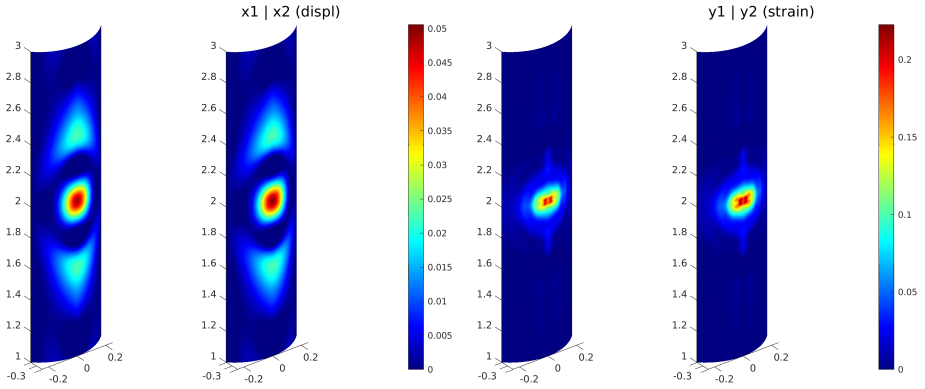

Fig. 7: 2 slightly different deformations and their associated quite different strain as obtained by the *Diff* model with 200 binary variables.

s of timeout, at 0.416 despite a very long timeout. The certified lower bound is not too far, at 0.025.

We did check that this worst-case found, displayed in Fig. 7 and which is not too far from the actual worst case that is known to be < 0.033, is coherent with the physical finite element model the DNN surrogate has been learned from. Hence, this brittleness of the surrogate is *not* a hallucination due to the learning process, but rather correctly reflects the physical system.

7 Conclusion

In this paper, we introduced a novel *Diff* MILP model for *global* verification questions, which is more accurate than the classical MILP model, even when using optimizations, namely the ITNE model of [28, 29] and VHAGaR [14]; see Table 1. We also explained how to support L_1-perturbations efficiently on top of classical L_∞-perturbations. Further, using simpler models, namely with the *2b+1* or the *1b* models, abstracting away some features of the *Diff* model, even better bounds can be reached for the complex questions of global robustness with L_1 bounds (Tables 3, 5). To further lower the bounds by $3-20$ times, we resort to PCA, concentrating the accuracy around the datasets, potentially being inaccurate for OOD inputs, over which the DNN is mostly inaccurate anyway, without sacrificing DNN accuracy over the dataset. This results into, to the best of our knowledge, the first *real-time* robustness certification reported for a DNN, with $\geq 70\%$ of robustness for an L_1-perturbation of ≥ 1 for different datasets. Still, global verification is only at its beginning, and combining different methods is necessary to scale to more complex DNNs.

References

1. Bak, S.: nnenum: Verification of ReLU Neural Networks with Optimized Abstraction Refinement. In: NASA Formal Methods Symposium. pp. 19–36. Springer (2021)
2. Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A.V., Criminisi, A.: Measuring Neural Net Robustness with Constraints. In: NeurIPS 2016 (2016), https://proceedings.neurips.cc/paper/2016/hash/980ecd059122ce2e50136bda65c25e07-Abstract.html
3. Brandon Paulsen, Jingbo Wang, C.W.: Reludiff: differential verification of deep neural networks. In: ICSE'20. pp. 714–726. ACM (2020)
4. 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) (2023)
5. Bunel, R., Lu, J., Turkaslan, I., Torr, P.H., Kohli, P., Kumar, M.P.: Branch and bound for piecewise linear neural network verification. Journal of Machine Learning Research **21**(42), 1–39 (2020)
6. Chen, Y., Wang, S., Qin, Y., Liao, X., Jana, S., Wagner, D.A.: Learning Security Classifiers with Verified Global Robustness Properties. In: CCS 2021 (2021), https://doi.org/10.1145/3460120.3484776
7. Chinesta, F., Huerta, A., Rozza, G., Willcox, K.: Model Reduction Methods, pp. 1–36. John Wiley & Sons (2017). https://doi.org/10.1002/9781119176817.ecm2110
8. De Conto, E., Genest, B., Easwaran, A.: Function+Data Flow: A Framework to Specify Machine Learning Pipelines for Digital Twinning. In: Adams, B., Zimmermann, T., Ozkaya, I., Lin, D., Zhang, J.M. (eds.) AIware'24. pp. 19–27. ACM (2024)
9. Durand, S., Lemesle, A., Chihani, Z., Urban, C., Terrier, F.: Reciph: Relational coefficients for input partitioning heuristic. In: 1st Workshop on Formal Verification of Machine Learning (WFVML 2022) (2022), https://pyrat-analyzer.com/
10. 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 (ICLR'22) (2022)
11. Fu, F., Wang, Z., Zhou, W., Wang, Y., Fan, J., Huang, C., Zhu, Q., Chen, X., Li, W.: REGLO: Provable Neural Network Repair for Global Robustness Properties. In: Wooldridge, M.J., Dy, J.G., Natarajan, S. (eds.) Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada. pp. 12061–12071. AAAI Press (2024), https://doi.org/10.1609/aaai.v38i11.29094
12. Gopinath, D., Katz, G., Pasareanu, C.S., Barrett., C.W.: DeepSafe: A Data-Driven Approach for Assessing Robustness of Neural Networks. In: ATVA 2018 (2018), https://doi.org/10.1007/978-3-030-01090-4_1
13. Huang, C., Fan, J., Chen, X., Li, W., Zhu, Q.: Divide and slide: Layer-wise refinement for output range analysis of deep neural networks. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **39**(11), 3323–3335 (2020). https://doi.org/10.1109/TCAD.2020.3013071
14. Kabaha, A., Cohen, D.D.: Verification of Neural Networks' Global Robustness. Proc. ACM Program. Lang. **8**(OOPSLA1) (2024). https://doi.org/10.1145/3649847, https://doi.org/10.1145/3649847

15. Karunanayake, N., Gunawardena, R., Seneviratne, S., Chawla, S.: Out-of-Distribution Data: An Acquaintance of Adversarial Examples - A Survey. ACM Comput. Surv. **57**(8), 210:1–210:40 (2025). https://doi.org/10.1145/3719292, https://doi.org/10.1145/3719292
16. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient smt solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) Computer Aided Verification. pp. 97–117. Cham (2017)
17. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., Dill, D.L., Kochenderfer, M.J., Barrett, C.: The Marabou Framework for Verification and Analysis of Deep Neural Networks. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification. pp. 443–452. Springer International Publishing, Cham (2019)
18. Leino, K., Wang, Z., Fredrikson, M.: Globally-Robust Neural Networks. In: ICML 2021 (2021), http://proceedings.mlr.press/v139/leino21a.html
19. Levy, N., Yerushalmi, R., Katz, G.: gRoMA: a Tool for Measuring Deep Neural Networks Global Robustness. In: arXiv.2301.02288 (2023), https://doi.org/10.48550/arXiv.2301.02288
20. Mangal, R., Nori, A.V., Orso, A.: Robustness of neural networks: a probabilistic and practical approach. In: ICSE 2019 (2019), https://doi.org/10.1109/ICSE-NIER.2019.00032
21. Muller, M.N., Makarchuk, G., Singh, G., Püschel, M., Vechev, M.: PRIMA: General and Precise Neural Network Certification via Scalable Convex Hull Approximations. In: Proc. ACM Program. Lang. POPL. vol. 6. Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3498704
22. Ruan, W., Wu, M., Sun, Y., Huang, X., Kroening, D., Kwiatkowska, M.: Global Robustness Evaluation of Deep Neural Networks with Provable Guarantees for the Hamming Distance. In: IJCAI 2019 (2019), https://doi.org/10.24963/IJCAI.2019/824
23. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An Abstract Domain for Certifying Neural Networks. Proc. ACM Program. Lang. **3**(POPL) (jan 2019). https://doi.org/10.1145/3290354
24. Sun, W., Lu, Y., Zhang, X., , Sun, M.: DeepGlobal: A framework for global robustness veri cation of feedforward neural networks. In J. Syst. Archit. (2022), https://doi.org/10.1016/J.SYSARC.2022.102582
25. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. International Conference on Learning Representations (ICLR'14) (2014)
26. Tjeng, V., Xiao, K., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. International Conference on Learning Representations (ICLR'19) (2019)
27. 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 Neural Network Robustness Verification. In: Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems. vol. 34, pp. 29909–29921. Curran Associates, Inc. (2021)
28. Wang, Z., Huang, C., Zhu, Q.: Efficient global robustness certification of neural networks via interleaving twin-network encoding. In: 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 1087–1092. IEEE (2022)
29. Wang, Z., Huang, C., Zhu, Q.: Efficient Global Robustness Certification of Neural Networks via Interleaving Twin-Network Encoding (Extended Abstract). In: Pro-

ceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China. pp. 6498–6503. ijcai.org (2023). https://doi.org/10.24963/IJCAI.2023/727, https://doi.org/10.24963/ijcai.2023/727
30. Wang, Z., Wang, Y., Fu, F., Jiao, R., Huang, C., Li, W., Zhu, Q.: A Tool for Neural Network Global Robustness Certification and Training. In: arXiv 2208.07289 (2022), https://doi.org/10.48550/arXiv.2208.07289
31. Wu, H., Isac, O., Zeljić, A., Tagomori, T., Daggitt, M., Kokke, W., Refaeli, I., Amir, G., Julian, K., Bassan, S., Huang, P., Lahav, O., Wu, M., Zhang, M., Komendantskaya, E., Katz, G., Barrett, C.: Marabou 2.0: A versatile formal analyzer of neural networks. In: Gurfinkel, A., Ganesh, V. (eds.) Proceedings of the 36^{th} International Conferenceon Computer Aided Verification (CAV '24). Lecture Notes in Computer Science, vol. 14681, pp. 249–264. Springer (Jul 2024), montreal, Canada
32. Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., Hsieh, C.: Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In: International Conference on Learning Representations (ICLR'21). OpenReview.net (2021)
33. Zhang, B., Jiang, D., He, D., Wang, L.: Rethinking Lipschitz Neural Networks and Certified Robustness: A Boolean Function Perspective. In: NeurIPS 2022 (2022), http://papers.nips.cc/paper_les/paper/2022/hash/7b04ec5f2b89d7f601382c422dfe07af-Abstract-Conference.html
34. 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: Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022 (2022)

A Proof of Proposition 2 and 3

Proposition 2. *Assuming $y_j \in [-\gamma_j, \gamma_j]$ and $x'_j, y_j + x'_j \in [\alpha_j, \beta_j]$, we have that $\hat{y}_j = \mathrm{ReLU}(x_j = y_j + x'_j) - \mathrm{ReLU}(x'_j)$ is the solution of the MILP program:*

$$\begin{aligned}
y_j + x'_j &\le a\beta_j & y_j + x'_j &\ge (1-a)\alpha_j \\
x'_j &\le a'\beta_j & x'_j &\ge (1-a')\alpha_j \\
\hat{y}_j &\le a\gamma_j & \hat{y}_j &\ge -a'\gamma_j \\
\hat{y}_j &\le y_j + x'_j - (1-a)\alpha_j & \hat{y}_j &\ge y_j + x'_j - a'\beta_j \\
\hat{y}_j &\le -x'_j + a\beta_j & \hat{y}_j &\ge -x'_j + (1-a')\alpha_j \\
\hat{y}_j &\le y_j + (1-a)\gamma_j & \hat{y}_j &\ge y_j - (1-a')\gamma_j
\end{aligned}$$

where $a, a' \in \{0, 1\}$ are binary variables, and a' is shared with the classical MILP encoding for $\hat{x}'_j = \mathrm{ReLU}(x'_j)$.

Proof (of Proposition 2). First, we list the inequalities:

$$\begin{aligned}
y_j + x'_j &\le a\beta_j & y_j + x'_j &\ge (1-a)\alpha_j \\
x'_j &\le a'\beta_j & x'_j &\ge (1-a')\alpha_j \\
\hat{y}_j &\le a\gamma_j & \hat{y}_j &\ge -a'\gamma_j \\
\hat{y}_j &\le y_j + x'_j - (1-a)\alpha_j & \hat{y}_j &\ge y_j + x'_j - a'\beta_j \\
\hat{y}_j &\le -x'_j + a\beta_j & \hat{y}_j &\ge -x'_j + (1-a')\alpha_j \\
\hat{y}_j &\le y_j + (1-a)\gamma_j & \hat{y}_j &\ge y_j - (1-a')\gamma_j
\end{aligned}$$

Follow the discuss in the proof of main text, we need to check the other inequalities of each cases.

$\mathbf{a} = \mathbf{0}, \mathbf{a}' = \mathbf{0}$: In this case, the inequalities become:

$$\begin{aligned}
y_j + x'_j &\le 0 & y_j + x'_j &\ge \alpha_j \\
x'_j &\le 0 & x'_j &\ge \alpha_j \\
\hat{y}_j &\le 0 & \hat{y}_j &\ge 0 \\
\hat{y}_j &\le y_j + x'_j - \alpha_j & \hat{y}_j &\ge y_j + x'_j \\
\hat{y}_j &\le -x'_j & \hat{y}_j &\ge -x'_j + \alpha_j \\
\hat{y}_j &\le y_j + \gamma_j & \hat{y}_j &\ge y_j - \gamma_j
\end{aligned}$$

Now we can see, $y_j + x'_j \le 0$ and $x'_j \le 0$ (lines 1&2), $\hat{y}_j = 0$ (line 3).

For other 3 lines, by above discussing and the assumption of the proposition that $y_j \in [-\gamma_j, \gamma_j]$ and $x'_j, y_j + x'_j \in [\alpha_j, \beta_j]$, they are implied by the following inequalities respectively (one for one)

$$\begin{aligned}
\hat{y}_j &\le 0 \le y_j + x'_j - \alpha_j & \hat{y}_j &\ge 0 \ge y_j + x'_j \\
\hat{y}_j &\le 0 \le -x'_j & \hat{y}_j &\ge 0 \ge -x'_j + \alpha_j \\
\hat{y}_j &\le 0 \le y_j + \gamma_j & \hat{y}_j &\ge 0 \ge y_j - \gamma_j
\end{aligned}$$

Therefore all 3 rest lines are implied by other inequalities.

$\mathbf{a} = \mathbf{1}, \mathbf{a}' = \mathbf{0}$**:** In this case, the inequalities become:

$$\begin{array}{ll} y_j + x'_j \le \beta_j & y_j + x'_j \ge 0 \\ x'_j \le 0 & x'_j \ge \alpha_j \\ \hat{y}_j \le \gamma_j & \hat{y}_j \ge 0 \\ \hat{y}_j \le y_j + x'_j & \hat{y}_j \ge y_j + x'_j \\ \hat{y}_j \le -x'_j + \beta_j & \hat{y}_j \ge -x'_j + \alpha_j \\ \hat{y}_j \le y_j & \hat{y}_j \ge y_j - \gamma_j \end{array}$$

Now we can see, $y_j + x'_j \ge 0$ and $x'_j \le 0$ (lines 1&2), hence $y_j \ge 0$; and $\hat{y}_j = y_j + x'_j$ (line 4).

By above discussing, we have:

$$\hat{y}_j \le y_j + x'_j \le y_j \quad \hat{y}_j \ge y_j + x'_j \ge 0$$

Then by the assumption of the proposition that $y_j \in [-\gamma_j, \gamma_j]$ and $x'_j, y_j + x'_j \in [\alpha_j, \beta_j]$, all 3 other lines are implied by the following inequalities respectively (one for one):

$$\begin{array}{ll} \hat{y}_j \le y_j \le \gamma_j & \hat{y}_j \ge y_j + x'_j \ge 0 \\ \hat{y}_j \le 0 \le -x'_j + \beta_j & \hat{y}_j \ge 0 \ge -x'_j + \alpha_j \\ \hat{y}_j \le y_j + x'_j \le y_j & \hat{y}_j \ge 0 \ge y_j - \gamma_j \end{array}$$

Therefore all 3 rest lines are implied by other inequalities.

$\mathbf{a} = \mathbf{0}, \mathbf{a}' = \mathbf{1}$**:** In this case, the inequalities become:

$$\begin{array}{ll} y_j + x'_j \le 0 & y_j + x'_j \ge \alpha_j \\ x'_j \le \beta_j & x'_j \ge 0 \\ \hat{y}_j \le 0 & \hat{y}_j \ge -\gamma_j \\ \hat{y}_j \le y_j + x'_j - \alpha_j & \hat{y}_j \ge y_j + x'_j - \beta_j \\ \hat{y}_j \le -x'_j & \hat{y}_j \ge -x'_j \\ \hat{y}_j \le y_j + \gamma_j & \hat{y}_j \ge y_j \end{array}$$

Now we can see, $y_j + x'_j \le 0$ and $x'_j \ge 0$ (lines 1&2), hence $y_j \le 0$; and $\hat{y}_j = -x'_j$ (line 5).

By above discussing, we have:

$$\hat{y}_j \le -x'_j \le 0 \qquad \hat{y}_j \ge -x'_j \ge -x'_j + (y_j + x'_j) \ge y_j$$

Then by the assumption of the proposition that $y_j \in [-\gamma_j, \gamma_j]$ and $x'_j, y_j + x'_j \in [\alpha_j, \beta_j]$, all 3 other lines are implied by the following inequalities respectively (one for one):

$$\begin{array}{ll} \hat{y}_j \le -x'_j \le 0 & \hat{y}_j \ge y_j \ge -\gamma_j \\ \hat{y}_j \le 0 \le y_j + x'_j - \alpha_j & \hat{y}_j \ge y_j \ge y_j + x'_j - \beta_j \\ \hat{y}_j \le 0 \le y_j + \gamma_j & \hat{y}_j \ge y_j \end{array}$$

Therefore all 3 rest lines are implied by other inequalities.

$\mathbf{a} = \mathbf{1}, \mathbf{a}' = \mathbf{1}$: In this case, the inequalities become:

$$\begin{array}{ll} y_j + x'_j \leq \beta_j & y_j + x'_j \geq 0 \\ x'_j \leq \beta_j & x'_j \geq 0 \\ \hat{y}_j \leq \gamma_j & \hat{y}_j \geq -\gamma_j \\ \hat{y}_j \leq y_j + x'_j & \hat{y}_j \geq y_j + x'_j - \beta_j \\ \hat{y}_j \leq -x'_j + \beta_j & \hat{y}_j \geq -x'_j \\ \hat{y}_j \leq y_j & \hat{y}_j \geq y_j \end{array}$$

Now we can see, $y_j + x'_j \geq 0$ and $x'_j \geq 0$ (lines 1&2) and $\hat{y}_j = y_j$ (line 6).

Then by the assumption of the proposition that $y_j \in [-\gamma_j, \gamma_j]$ and $x'_j, y_j + x'_j \in [\alpha_j, \beta_j]$, all 3 other lines are implied by the following inequalities respectively (one for one):

$$\begin{array}{ll} \hat{y}_j \leq y_j \leq \gamma_j & \hat{y}_j \geq y_j \geq -\gamma_j \\ \hat{y}_j \leq y_j \leq y_j + x'_j & \hat{y}_j \geq y_j \geq y_j + x'_j - \beta_j \\ \hat{y}_j \leq -x'_j + (y_j + x'_j) \leq -x'_j + \beta_j & \hat{y}_j \geq y_j - (y_j + x'_j) \geq -x'_j \end{array}$$

Therefore all 3 rest lines are implied by other inequalities.

□

Proposition 3. *Given $y_i \in [-\gamma_i, \gamma_i]$, $\hat{y}_i$ is a solution of the MILP program :*

$$\begin{array}{ll} \hat{y}_i \leq a\gamma_i & \hat{y}_i \geq y_i - a\gamma_i \\ \hat{y}_i \geq (a-1)\gamma_i & \hat{y}_i \leq y_i + (1-a)\gamma_i \end{array}$$

where $a \in \{0, 1\}$ is a binary variable.

Proof (of Proposition 3). First, we list the inequalities:

$$\begin{array}{ll} \hat{y}_i \leq a\gamma_i & \hat{y}_i \geq y_i - a\gamma_i \\ \hat{y}_i \geq (a-1)\gamma_i & \hat{y}_i \leq y_i + (1-a)\gamma_i \end{array}$$

Similar to Proposition 2, we consider by cases.

$\mathbf{a} = \mathbf{0}$: In this case, the inequalities become:

$$\begin{array}{ll} \hat{y}_i \leq 0 & \hat{y}_i \geq y_i \\ \hat{y}_i \geq -\gamma_i & \hat{y}_i \leq y_i + \gamma_i \end{array}$$

Now we can see, $y_i \leq 0$ (line 1) and $y_i \leq \hat{y}_i \leq 0$. Line 2 is implied by other inequalities since

$$\hat{y}_i \geq y_i \geq -\gamma_i \qquad \hat{y}_i \leq 0 \leq y_i + \gamma_i$$

$\mathbf{a} = \mathbf{1}$: In this case, the inequalities become:

$$\hat{y}_i \leq \gamma_i \qquad \hat{y}_i \geq y_i - \gamma_i$$
$$\hat{y}_i \geq 0 \qquad \hat{y}_i \leq y_i$$

Now we can see, $y_i \geq 0$ (line 2) and $0 \leq \hat{y}_i \leq y_i$. Line 1 is implied by other inequalities since

$$\hat{y}_i \leq y_i \leq \gamma_i \qquad \hat{y}_i \geq 0 \geq y_i - \gamma_i$$

Now we finish the proof. □