HYBRID MILP TO EFFICIENTLY AND ACCURATLY SOLVE HARD DNN VERIFICATION INSTANCES

Anonymous authors
Paper under double-blind review

ABSTRACT

α, β-CROWN has won the last 4 VNNcomp(etitions), as the DNN verifier with the best trade-off between accuracy and runtime. VNNcomp however is focusing on relatively easy verification instances. In this paper, we consider *harder* verification instances, on which α, β-Crown displays a large number ($20-58\%$) of undecided instances, that is, instances that can neither be verified, nor an explicit attack can be found. Enabling larger time-outs for α, β-Crown only improves verification rate by few percents, leaving a large gap of undecided instances while already taking a considerable amount of time. Resorting to other techniques, such as complete verifiers, does not fare better even with very large time-outs: They would theoretically be able to close the gap, but with an untractable runtime on all but small *hard* instances.

In this paper, we study the MILP encoding of ReLU-DNNs, and provide new insights in the LP relaxation. This allows us to carefully craft a *partial MILP* solution which selects automatically few neurons encoded as integer variables, the rest using the LP relaxation. Compared with previous attempts, we can reduce the number of integer variables by around 4 times while maintaining the same level of accuracy. Implemented in *Hybrid MILP*, calling first α, β-Crown with a short time-out to solve easier instances, and then partial MILP for those for which α, β-Crown fails, produces a very accurate yet efficient verifier, reducing tremendously the number of undecided instances ($8-15\%$), while keeping a reasonable runtime ($46s - 417s$ on average per instance).

1 INTRODUCTION

Deep neural networks (DNNs for short) have demonstrated remarkable capabilities, achieving human-like or even superior performance across a wide range of tasks. However, their robustness is often compromised by their susceptibility to input perturbations Szegedy et al. (2014). This vulnerability has catalyzed the verification community to develop various methodologies, each presenting a unique balance between completeness and computational efficiency Katz et al. (2019; 2017); Singh et al. (2019b). This surge in innovation has also led to the inception of competitions such as VNNComp Brix et al. (2023b), which aim to systematically evaluate the performance of neural network verification tools. While the verification engines are generic, the benchmarks usually focus on local robustness, i.e. given a DNN, an image and a small neighbourhood around this image, is it the case that all the images in the neighbourhood are classified in the same way. For the past 5 years, VNNcomp has focused on rather easy instances, that can be solved within tens of seconds (the typical hard time-out is 300s). For this reason, DNN verifiers in the past years have mainly focused on optimizing for such easy instances. Among them, NNenum Bak (2021), Marabou Katz et al. (2019); Wu et al. (2024), and PyRAT Durand et al. (2022), respectively 4th, 3rd and 2sd of the last VNNcomp'24 Brix et al. (2024) and 5th, 2sd and 3rd of the VNNcomp'23 Brix et al. (2023a); MnBAB Ferrari et al. (2022), 2sd in VNNcomp'22 Müller et al. (2022), built upon ERAN Singh et al. (2019b) and PRIMA Müller et al. (2022); and importantly, α, β-Crown Wang et al. (2021); Xu et al. (2021), the winner of the last 4 VNNcomp, benefiting from branch-and-bound based methodology Zhang et al. (2022); Bunel et al. (2020). We will thus focus in the following mostly on α, β-Crown.

Easy instances does not mean small DNNs: for instance, a ResNet architecture for CIFAR10 (with tens of thousands of neurons) has been fully checked by α, β-Crown Wang et al. (2021), each instance taking only a couple of seconds to either certify that there is no robustness attack, or finding a

Network Perturbation	Accuracy	Upper Bound	α, β-Crown TO=10s	α, β-Crown TO=30s	α, β-Crown TO=2000s
MNIST 5$\times$100 $\epsilon = 0.026$	99%	90%	33% 6.9s	35% 18.9s	40% 1026s
MNIST 5$\times$200 $\epsilon = 0.015$	99%	96%	46% 6.5s	49% 16.6s	50% 930s
MNIST 8$\times$100 $\epsilon = 0.026$	97%	86%	23% 7.2s	28% 20.1s	28% 930s
MNIST 8$\times$200 $\epsilon = 0.015$	97%	91%	35% 6.8s	36% 18.2s	37% 1083s
MNIST 6$\times$500 $\epsilon = 0.035$	100%	94%	41% 6.4s	43% 16.4s	44% 1003s
CIFAR CNN-B-adv $\epsilon = 2/255$	78%	62%	34% 4.3s	40% 8.7s	42% 373s
CIFAR ResNet $\epsilon = 2/255$	29%	25%	25% 2s	25% 2s	25% 2s

Table 1: Images verified by α, β-Crown with different time-outs (TO) on 7 DNNs, and average runtime per image. The 6 first DNNs are hard instances. The last DNN (ResNet) is an easy instance (trained using Wong to be easy to verify, but with a very low accuracy level), provided for reference.

very close neighbour with a different decision. One issue is however that easy instances are trained specifically to be easier to verify e.g. using DiffAI Mirman et al. (2018) PGD Madry et al. (2018), which can impact the accuracy of the network, i.e. answering correctly to an unperturbed input. For instance, this ResNet was trained using Wong, and only 29% of its answers are correct (the other 71% are thus not even tested by α, β-Crown). While more accurate trainers for verification have been recently developed Xu et al. (2024), they can only simplify one given verification specification by a limited amount before hurting accuracy, turning e.g. very hard verification instances into hard verification instances. Also, verification questions intrinsically harder than local robustness, such as bounding on Lipschitz constants Wang et al. (2022) globally or asking several specification at once, makes the instance particularly harder. Last, there are many situations (workflow, no access to the dataset...) where using specific trainers to learn easy to verify DNN is simply not possible, leading to *verification-agnostic* networks Dathathri et al. (2020). The bottom line is, one cannot expect only *easy* verification instances: *hard* verification instances need to be explored as well.

In this paper, we focused on the 6 *hard* ReLU-DNNs that have been previously tested in Wang et al. (2021), which display a large gap ($\geq$ 20%) between images that can be certified by α, β-Crown and the upper bound when we remove those which can be falsified. In turns, hard instances does not necessarily mean very large DNNs, the smallest of these hard DNNs having only 500 hidden neurons, namely MNIST 5$\times$100. We first dwelve into the scaling of α, β-Crown, to understand how longer Time-Out (TO) affects the number of undecided images and the runtime. Table 1 reveals that even allowing for 200 times longer time outs only improves the verification from 2% to 8%, leaving a considerable $20\% - 50\%$ gap of undecided images, while necessitating vastly longer runtime (300s-1000s in average per instance).

The size of the smallest DNN (500 hidden neurons) makes it believable to be solved by complete verifiers such as Marabou 2.0, NNenum or a Full MILP encoding. While they should theoretically be able to close the gap of undecided images, in practice, even with a large 10 000s Time-out, Table 2 reveals that only NNenum succeeds to verify images not verified by α, β-Crown, limited to 9% more images out of the 50% undecided images, and with a very large runtime of almost 5000s per image. It seemed pointless to test complete verifiers on larger networks.

Network	Accuracy	Upper	Marabou 2.0	NNenum	Full MILP
MNIST 5$\times$100 $\epsilon = 0.026$	99%	90%	28% 6200s	49% 4995s	40 % 6100s

Table 2: Result of complete verifiers on the hard 5x100 with TO = 10 000s. Complete verifier barely (9% out of 50%) outperform α, β-Crown (40%, 1026s), despite much larger runtime.

Our three main contributions address the challenges of verifying *hard* DNNs efficiently:

1. Our first contribution studies the *LP relaxation* of the exact MILP encoding of ReLUs. We establish in Proposition 1 its equivalence with the so-called ”triangular abstraction”.
2. It allows us to choose through a novel Utility function few neurons to encode with the exact MILP encoding, while others are treated with the efficient LP relaxation, giving rise to partial MILP (pMILP). Specifically, the novelty of Utility resides in the use of the solution to an (efficient) full LP call on the node z we want to bound. Utility can then precisely evaluate how much accuracy is gained by switching neuron a from LP (solution of the LP call) to the exact MILP encoding of ReLU (exact computation from the solution, which can be made thanks to Proposition 1), with a proved bound on the precision (Proposition 2). pMILP is much more efficient than previous attempts Huang et al. (2020), necessitating ≈ 4 times less integer variables for the same accuracy (Table 3), because Utility focuses on the solution local to the input, rather than on the DNN. To the best of our knowledge, this is the first time such a solution of an (LP) call is used to evaluate the contribution of each neuron, including heuristics for BaB, e.g. Ferrari et al. (2022); Wang et al. (2021).
3. We then propose a new verifier, called *Hybrid MILP*, invoking first α, β-Crown with short time-out to settle the easy instances. On those (*hard*) instances which are neither certified nor falsified, we call pMILP with few neurons encoded as integer variables. Experimental evaluation reveals that Hybrid MILP achieves a beneficial balance between accuracy and completeness compared to prevailing methods. It reduces the proportion of undecided inputs from $20 - 58\%$ (α, β-Crown with 2000s TO) to $8 - 15\%$, while taking a reasonable average time per instance ($46 - 420$s), Table 4. It scales to fairly large networks such as CIFAR-10 CNN-B-Adv Dathathri et al. (2020), with more than 20 000 neurons.

Limitation: We consider DNNs employing the standard ReLU activation function, though our findings should extend to other activation functions, following similar extention by Huang et al. (2020).

1.1 Related Work

We compare Hybrid MILP with major verification tools for DNNs to clarify our methodology and its distinction from the existing state-of-the-art. It scales while preserving good accuracy, through targeting a limited number of binary variables, stricking a good balance between exact encoding of a DNN using MILP Tjeng et al. (2019) (too slow) and LP relaxation (too inaccurate). MIP-planet Ehlers (2017) opts for a different selection of binary variables, and execute one large MILP encoding instead of Hybrid MILP's many small encodings, which significantly reduce the number of binary variables necessary for each encoding. In Huang et al. (2020), small encodings are also considered, however with a straightforward choice of binary nodes based on the weight of outgoing edges, which need much more integer variables (thus runtime) to reach the same accuracy.

Hybrid MILP can be seen as a refinement of α, β-Crown Wang et al. (2021), though its refined accurate path is vastly different than the base Branch and Bound technique used in α, β CROWN, BaBSR Bunel et al. (2020) and MN-BaB Ferrari et al. (2022), which call BaB once per output neuron. In the worst case, this involves considering all possible ReLU configurations, though branch and bound typically circumvents most possibilities. In simple networks, like those trained robustly, branch and bound is highly efficient, focusing on branches crucial for verifying the actual property. However, branch and bound hits a complexity barrier when verifying harder instances, due to an overwhelming number of branches, as displayed in Table 1. This is not the case of Hybrid MILP, see Table 4, which is much more accurate than α, β-Crown. That shortcoming for hard instances was witnessed in Wang et al. (2021), and a very specific solution using the full MILP encoding for the first few layers of a DNN was drafted, following similar proposal Singh et al. (2019c). The main issue is that it is slow and it cannot scale to DNNs with many neurons, as every neurons are encoded using an integer variable, making it not that accurate for intermediate networks (e.g. 9×100, 9×200, Table 4), and not usable for larger DNNs (6×500, CNN-B-Adv), whereas Hybrid MILP does scale.

Last, ERAN-DeepPoly Singh et al. (2019b) computes bounds on values very quickly, by abstracting the weight of every node using two functions: an upper function and a lower function. While the upper function is fixed, the lower function offers two choices. It relates to the LP encoding through the following new (to our knowledge) insight: Proposition 1 state that the LP relaxation precisely matches the intersection of these two choices. Consequently, LP is more accurate (but slower) than DeepPoly, and Hybrid MILP is considerably more precise. Regarding PRIMA Müller et al. (2022), the approach involves explicitly maintaining dependencies between neurons.

Finally, methods such as Reluplex / Marabou Katz et al. (2017; 2019) abstract the network: they diverge significantly from those abstracting values such as PRIMA, α, β-CROWN)Müller et al. (2022); Wang et al. (2021), Hybrid MILP. These network-abstraction algorithms are designed to be *complete* but completeness comes at the price of significant scalability challenges, and in practice they time-out on hard instances as shown in Table 2.

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. To simplify the notations, we restrict the presentation to feed-forward, fully connected ReLU Deep Neural Networks (DNN for short), where the ReLU function is $ReLU : \mathbb{R} \to \mathbb{R}$ with $ReLU(x) = x$ for $x \geq 0$ and $ReLU(x) = 0$ 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

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

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 call 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. Finally, pre- and post-activation neurons are called *nodes*, and when we refer to a specific node/neuron, we use a, b, c, d, n to denote them, and $W_{a,b} \in \mathbb{R}$ to denote the weight from neuron a to b. Similarly, for input $\boldsymbol{x}$, we denote by $\text{value}_{\boldsymbol{x}}(a)$ the value of neuron a when the input is $\boldsymbol{x}$. A path π is a sequence $\pi = (a_i)_{k \leq i \leq k'}$ of neurons in consecutive layers, and the weight of π is $weight(\pi) = W_{a_k, a_{k+1}} \times \cdots \times W_{a_{k'-1}, a_{k'}}$.

Concerning the verification problem, we focus on the well studied local-robustness question. Local robustness asks to determine whether the output of a neural network will be affected under small perturbations to the input. Formally, for an input $\boldsymbol{x}$ perturbed by $\varepsilon > 0$ under distance d, then the DNN is locally ε-robust in $\boldsymbol{x}$ whenever:

$$\forall \boldsymbol{x'} \text{ s.t. } d(\boldsymbol{x}, \boldsymbol{x'}) \leq \varepsilon, \text{ we have } argmax_i(f(\boldsymbol{x'})[i]) = argmax_i(f(\boldsymbol{x})[i])$$

3 Value Abstraction for DNN verification

In this section, we describe different value (over-)abstractions on $\boldsymbol{z}$ that are used by efficient algorithms to certify robustness around an input $\boldsymbol{x}$. Over-abstractions of values include all values for $\boldsymbol{z}$ in the neighbourhood of $\boldsymbol{x}$, and thus a certificate for safety in the over-abstraction is a proof of safety for the original input $\boldsymbol{x}$.

3.1 The Box Abstractions

The concept of value abstraction involves calculating upper and lower bounds for the values of certain neurons in a Deep Neural Network (DNN) when inputs fall within a specified range. This approach aims to assess the network's robustness without precisely computing the values for every input within that range.

Firstly, it's important to note that weighted sums represent a linear function, which can be explicitly expressed with relative ease. However, the ReLU (Rectified Linear Unit) function presents a challenge in terms of accurate representation. Although ReLU is a relatively straightforward piece-wise linear function with two modes (one for $x < 0$ and another for $x \geq 0$), it is not linear. The

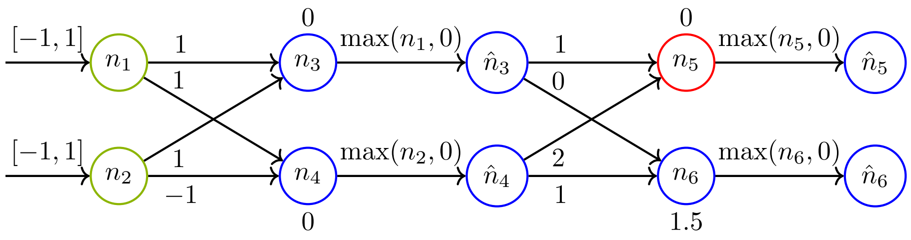

Figure 1: A DNN. Every neuron is separated into 2 nodes, n pre- and $\hat{n}$ post-ReLU activation.

complexity arises when considering the compounded effects of the ReLU function across the various layers of a ReLU DNN. It's worth noting that representing $\mathrm{ReLU}(x)$ precisely is feasible when x is "*stable*", meaning it's consistently positive or consistently negative, as there's only one linear mode involved in each scenario. Consequently, the primary challenge lies in addressing "*unstable*" neurons, where the linearity of the function does not hold consistently.

Consider the simpler abstraction, termed "Box abstraction" Singh et al. (2019b): it inductively computes the bounds for each neuron in the subsequent layer independently. This is achieved by considering the weighted sum of the bounds from the previous layer, followed by clipping the lower bound at $\max(0, \text{lower bound})$ to represent the ReLU function, and so forth. For all $i \geq 3$, define $x_i = \text{value}_{\boldsymbol{x}}(n_i)$, where $\boldsymbol{x} = (x_1, x_2)$. Taking the DNN example from Fig 1, assume $x_1, x_2 \in [-1, 1]$. This implies that $x_3, x_4 \in [-2, 2]$. After applying the ReLU function, $\hat{x}_3, \hat{x}_4$ are constrained to $[0, 2]$, leading to $x_5 \in [0, 6]$ and $x_6 \in [0, 2]$. The bounds for $n_1, \ldots, n_4$ are exact, meaning for every α within the range, an input $\boldsymbol{y}$ can be found such that $\text{value}_{\boldsymbol{y}}(n_i) = \alpha$. However, this precision is lost from the next layer (beginning with n_5, n_6) due to potential dependencies among preceding neurons. For example, it is impossible for $x_5 = \text{value}_{\boldsymbol{x}}(n_5)$ to reach 6, as it would necessitate both $x_3 = 2$ and $x_4 = 2$, which is not possible at the same time as $x_3 = 2$ implies $x_1 = x_2 = 1$ and $x_4 = 2$ implies $x_2 = -1$ (and $x_1 = 1$), a contradiction.

3.2 MILP, LP AND PARTIAL MILP ENCODINGS FOR DNNS

At the other end of the spectrum, we find the Mixed Integer Linear Programming (MILP) value abstraction, which is a complete (but inefficient) method. Consider an unstable neuron n, whose value $x \in [\mathrm{LB}(n), \mathrm{UB}(n)]$ 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 integer (actually even binary) variable a valued in $\{0, 1\}$, using constants $\mathrm{UB}(n), \mathrm{LB}(n)$ with 4 constraints Tjeng et al. (2019):

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

For all $x \in [\mathrm{LB}(n), \mathrm{UB}(n)] \setminus 0$, there exists a unique solution $(a, \hat{x})$ that meets these constraints, with $\hat{x} = \mathrm{ReLU}(x)$ Tjeng et al. (2019). The value of a is 0 if $x < 0$, and 1 if $x > 0$, and can be either if $x = 0$. This encoding approach can be applied to every (unstable) ReLU node, and optimizing its value can help getting more accurate bounds. However, for networks with hundreds of *unstable* nodes, the resulting MILP formulation will contain numerous integer variables and generally bounds obtained will not be accurate, even using powerful commercial solvers such as Gurobi.

MILP instances can be linearly relaxed into LP over-abstraction, where variables originally restricted to integers in $\{0, 1\}$ (binary) are relaxed to real numbers in the interval $[0, 1]$, while maintaining the same encoding. As solving LP instances is 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 abstraction*.

In this paper, we propose to use *partial MILP*, to get interesting trade-offs between accuracy and runtime: for a set of unstable neurons X, we denote by MILP_X the MILP encoding where variables encoding X are binary, and other variables are linear variables using the LP relaxation. We say that nodes in X are *opened*. To further limit the number of binary variables needed for a given accuracy, we devise the same iterative approach as the box abstraction or DeepPoly Singh et al.

(2019b), computing lower and upper bounds $\mathrm{LB}(n), \mathrm{UB}(n)$ for neurons n of a layer, that are used when computing values of the next layer, thus necessitating less variables from previous layers.

The crucial factor in such an approach is to *select* few opened neurons in X which are the most important for the accuracy. An extreme strategy was adopted in Huang et al. (2020), where only nodes of the immediate previous layer could be opened, and the measure to choose a neuron a to open to compute the values for neuron b was to consider $|W_{ab}|(\mathrm{UB}(a) - \mathrm{LB}(a))$. To obtain a more accurate measure, that can also scale to deeper layers and is not limited to open nodes from the immediate previous layer, we first study the LP abstraction.

Consider an unstable neuron n, that is $\mathrm{LB}(n) < 0 < \mathrm{UB}(n)$. Let x represent the input value of n, hence with $\mathrm{LB}(n) < x < \mathrm{UB}(n)$, and $\hat{x}$ representing $\mathrm{ReLU}(x)$. We have:

Proposition 1. *The LP relaxation on $\hat{x}$ is equivalent with:*

$$\mathrm{ReLU}(x) \leq \hat{x} \leq \mathrm{UB}(n)\frac{x - \mathrm{LB}(n)}{\mathrm{UB}(n) - \mathrm{LB}(n)} \tag{1}$$

Proof. The lower bound on $\hat{x}$ is simple, as $\hat{x} \geq 0 \wedge \hat{x} \geq x$ is immediatly equivalent with $\hat{x} \geq \mathrm{ReLU}(x)$.

We now show that the three constraints $\hat{x} \leq \mathrm{UB}(n) \cdot a \wedge \hat{x} \leq x - \mathrm{LB}(n) \cdot (1-a) \wedge a \in [0,1]$ translates into $\hat{x} \leq \mathrm{UB}(n)\frac{x-\mathrm{LB}(n)}{\mathrm{UB}(n)-\mathrm{LB}(n)}$. We have $\hat{x}$ is upper bounded by $max_{a\in[0,1]}(min(\mathrm{UB}(n)\cdot a, x - \mathrm{LB}(n)(1-a)))$, and this bound can be reached. Furthermore, using standard function analysis tools (derivative...), we can show that the function $a \mapsto \min(\mathrm{UB}(n)\cdot a, x - \mathrm{LB}(n)(1-a))$ attains its maximum when $\mathrm{UB}(n)\cdot a = x - \mathrm{LB}(n)(1-a)$, leading to the equation $(\mathrm{UB}(n) - \mathrm{UB}(n))a = x - \mathrm{LB}(n)$ and consequently $a = \frac{x-\mathrm{LB}(n)}{\mathrm{UB}(n)-\mathrm{LB}(n)}$. This results in an upper bound $\hat{x} \leq \mathrm{UB}(n)\frac{x-\mathrm{LB}(n)}{\mathrm{UB}(n)-\mathrm{LB}(n)}$, which can be reached, hence the equivalence. □

Notice that this is close to the DeepPoly abstractions Singh et al. (2019b), but LP simultaneous considers the two linear lower bounding functions for the ReLU operation, namely $\mathrm{ReLU}(x) \geq 0$ and $\mathrm{ReLU}(x) \geq x$, while the DeepPoly abstraction restricts itself to the application of a single one of these linear bounding functions.

4 Important neurons for accuracy and Utility function.

In this section, we evaluate how each neuron would impact the accuracy of encoding this neuron with an integer (actually binary) or a linear variable. We will say we open a neuron when this neuron is selected with a binary variable. We denote by X a choice of set of open neurons, and by $\mathcal{M}_X$ the MILP model where variables from X are encoded with binary variables, and other variables are using the LP linear relaxation.

Let z be a neuron of the DNN we want to evaluate, for instance computing an upper bound on the value it can take. We denote by $n < z$ the fact that a neuron $n$appears on a layer before the layer of z, and $n \leq z$ if it can also be z. Let us denote $(\mathrm{Sol\_max}_X^z(n))_{n\leq z}$ a solution of $\mathcal{M}_X$ maximizing z. In particular, $\mathrm{Sol\_max}_X^z(z)$ is the maximum of z under $\mathcal{M}_X$.

Consider $(\mathrm{Sol\_max}_\emptyset^z(n))_{n\leq z}$ (this can be obtained very efficiently by *one* call to an LP solver). An accurate utility function is to evaluate $\mathrm{Improve\_max}^z(a) = \mathrm{Sol\_max}_\emptyset^z(z) -$ $\mathrm{Sol\_max}_{\{a\}}^z(z)$, representing how much opening neuron $a < z$ reduces the maximum computed for z compared with using only LP. We have $\mathrm{Improve\_max}^z(a) \geq 0$ as $\mathrm{Sol\_max}_{\{a\}}^z$ fullfils all the constraints of $\mathcal{M}_\emptyset$, so $\mathrm{Sol\_max}_{\{a\}}^z(z) \leq \mathrm{Sol\_max}_\emptyset^z(z)$. Similarly, we define $(\mathrm{Sol\_min}_\emptyset^z(n))_{n\leq z}$ and $\mathrm{Improve\_min}^z(a)$. Calling MILP on $\mathcal{M}_{\{a\}}$ for each neuron $a \leq z$ would however be very time consuming when the number of neurons a to evaluate is large. Instead, we focus on the following upper bounding of $\mathrm{Improve\_max}^z(a)$.

For all neurons n, let $\mathrm{sol}(n)$ $=\mathrm{Sol\_max}_\emptyset^z(n)$ be the value of neuron a in the solution of the LP instance $\mathrm{Sol\_max}_\emptyset^z$ to maximize z. To estimate $\mathrm{Improve\_max}^z(a)$, we define $\mathrm{Utility\_max}^z(a)$, first for neurons a one layer before z, by computing by how much the value of z will change if a is opened and other values remain the same - in particular, $value(\hat{a}) = \mathrm{ReLU}(sol(a))$. We define:

$$\text{Utility\_max}^z(a) = W_{az} \times (\text{sol}(\hat{a}) - \text{ReLU}(\text{sol}(a)))$$

In the case $W_{az} < 0$, to maximize z, the LP engine sets $\text{sol}(\hat{a})$ to its minimal value, which is $\text{sol}(\hat{a}) = \text{ReLU}(\text{sol}(a))$ thanks to Prop. 1. In this case, we have $\text{Utility\_max}^z(a) = 0$. Thus, we have $\text{Utility\_max}^z(a) \geq 0$ in all cases as $\text{sol}(\hat{a}) \geq \text{ReLU}(\text{sol}(a))$ by Prop. 1. We will show with a more general definition that $0 \leq \text{Improve\_max}^z(a) \leq \text{Utility\_max}^z(a)$ in Prop. 2. Thus, $\text{Utility\_max}^z(a)$ can be used to approximate $\text{Improve\_max}^z(a)$. In particular, for all nodes a with $W_{az} < 0$, this node will have the smallest $\text{Utility\_max}^z(a) = 0$ (thus will not get picked in the open nodes X), and indeed it is not having any impact on $\text{Sol\_max}^z_{\{a\}}(z)$. This is one stricking difference (but not the only one) with choosing utility based on $|W_{az}|$ Huang et al. (2020).

Now, consider neuron a a two layers before z. We use b to denote neurons in the layer ℓ between a and z. If we open a without changing its value $\text{sol}(a)$, then the change $\Delta(\hat{a})$ in the weight of $\hat{a}$ is $\Delta(\hat{a}) = \text{ReLU}(\text{sol}(a)) - \text{sol}(\hat{a}) \leq 0$ as above. Its impact on z is no more direct with W_{az}, but it is through ℓ. We let $\Delta(b) = W_{ab}\Delta(\hat{a})$ for all $b \in \ell$. Based on Proposition 1, we can evaluate the impact $\Delta(\hat{b})$ of opening a on the value of each $\hat{b}$, by using the upper and lower bound $\text{UB}(b), \text{LB}(b)$:

$$\Delta(\hat{b}) = \begin{cases} \frac{\text{UB}(b)}{\text{UB}(b)-\text{LB}(b)}\Delta(b), & \text{if } W_{bz} > 0 \\ \max(\Delta(b), -\text{sol}(b)), & \text{if } W_{bz} < 0 \text{ and } \text{sol}(b) \geq 0 \\ \max(\Delta(b) + \text{sol}(b), 0), & \text{if } W_{bz} < 0 \text{ and } \text{sol}(b) < 0 \end{cases}$$

Indeed, if $W_{bz} > 0$, then according to Proposition 1, the LP solver sets $\text{sol}(\hat{b}) = \text{sol}(b)\frac{\text{UB}(b)}{\text{UB}(b)-\text{LB}(b)} +$ Cst to maximize z. Changing b by $\Delta(b)$ thus results in changing $\text{sol}(\hat{b})$ by $\frac{\text{UB}(b)}{\text{UB}(b)-\text{LB}(b)}\Delta(b)$. If $W_{bz} \leq 0$, then the LP solver sets $\text{sol}(\hat{b})$ to the lowest possible value to maximize z, which happens to be $\text{ReLU}(b)$ according to Proposition 1. If $\text{sol}(b) < 0$, then we have $\text{sol}(\hat{b}) = \text{ReLU}(b) = 0$ and opening a change the 0 value only if $\text{sol}(b) + \Delta(b) > 0$. If $\text{sol}(b) > 0$, then $\text{sol}(\hat{b}) = \text{ReLU}(\text{sol}(b)) = \text{sol}(b)$, and the change to $\hat{b}$ will be the full $\Delta(b)$, unless $\Delta(b) < -\text{sol}(b) < 0$ in which case it is $-\text{sol}(b)$. We then set:

$$\text{Utility\_max}^z(a) = -\sum_{b \in \ell} W_{bz}\Delta(\hat{b})$$

Proposition 2. $0 \leq \text{Improve\_max}^z(a) \leq \text{Utility\_max}^z(a)$.

Proof. Consider $\text{sol}'(n))_{n \leq z})$ with $\text{sol}'(n) = \text{sol}(n)$ for all $n \notin \{z, \hat{a}\} \cup \{b, \hat{b} \mid b \in \ell\}$. In particular, $\text{sol}'(a) = \text{sol}(a)$. Now, define $\text{sol}'(\hat{a}) = \text{ReLU}(\text{sol}(a))$. That is, $\text{sol}'(\hat{a})$ is the correct value for $\hat{a}$, obtained if we open neuron a, compared to the LP abstraction for $\text{sol}(\hat{a})$. We also define $\text{sol}'(b) = \text{sol}(b) + \Delta(b)$ and $\text{sol}'(\hat{b}) = \text{sol}(\hat{b}) + \Delta(\hat{b})$. Last, $\text{sol}'(z) = \text{sol}(z) + \sum_{b \in \ell} W_{bz}\Delta(\hat{b})$.

It is easy to check that $(\text{sol}'(n))_{n \leq z}$ satisfies the constraints in $\mathcal{M}_{\{a\}}$, as opening a changes the value of $\hat{a}$ from $\text{sol}(\hat{a})$ to $\text{ReLU}(\text{sol}(a))$, and the contribution from a to $b, \hat{b}, z$ are respected. As $\text{sol}'(z)$ is a solution of $\mathcal{M}_{\{a\}}$, it is smaller or equal to the maximal solution: $\text{sol}'(z) \leq \text{Sol\_max}^z_{\{a\}}(z)$. That is, $\text{sol}(z) - \text{sol}'(z) \geq \text{sol}(z) -$ $\text{Sol\_max}^z_{\{a\}}(z)$, i.e. $\text{Utility\_max}^z(a) \geq \text{Improve\_max}^z(a)$. In particular, we have that $\text{Utility\_max}^z(a) \geq 0$, which was not obvious from the definition. □

We can proceed inductively in the same way to define $\text{Utility\_max}^z(a)$ for deeper neurons a.

5 Experimental Evaluation

We implemented Hybrid MILP in Python 3.8, and Gurobi 9.52 was used for solving LP and MILP problems. We conducted our evaluation on an AMD Threadripper 7970X (32 cores@4.0GHz, 5nm) with 256 GB of main memory and 2 NVIDIA RTX 4090.

The objectives of our evaluation was to answer the following questions:

1. How does the the choice of the set X impacts the accuracy of MILP$_X$?
2. How accurate is Hybrid MILP, and how efficient is it?

5.1 Evaluation of the Utility function to choose neurons to open

To measure the impact of the utility function to select neurons to open, we focused on a small hard DNN, namely 5×100, so as to be able to compute exact bounds for the first few layers using a full MILP encoding of the DNN for comparison purpose. We tested over the $\boldsymbol{x} = 59$th image in the MNIST dataset, as it has a large number of unstable ReLU nodes in the first few layers (61 in the first and 55 in the second layer), so we can experiment with a larger choice of values) for $K = |X|$ the size of set X. To measure the accuracy, we measure the uncertainty of all nodes in a layer: the uncertainty of a node is the range between its computed lower and upper bound. We then average the uncertainty among all the nodes of the layer. Formally, the uncertainty of a node a with bounds $[\mathrm{LB}, \mathrm{UB}]$ is uncert$(a) = \mathrm{UB} - \mathrm{LB}$. The average uncertainty of layer ℓ is $\dfrac{\sum_{a \in l} \text{uncert}(a)}{size(\ell)}$.

We focus on the uncertainty of nodes in the third layers, wrt ReLU nodes in the first and second layer. The bounds for nodes of the first two layers are computed exactly using the full MILP encoding. We report in Table 3 the average uncertainty of MILP$_X$ following the choice of the K heaviest neurons for our **Utility** function, compared with a random choice, both for nodes exclusively from the previous Layer 2 or from both Layers 1 and 2. We compared with choosing based on strength$(n) = (\mathrm{UB}(n) - \mathrm{LB}(n)) \cdot |W_{nz}|$ Huang et al. (2020), which is limited to the previous Layer 2.

$\|X\|$	$X \subseteq$ Layer 2, max = 55			$X \subseteq$ Layers 1&2, max = 116	
	Random	Huang et al. (2020)	**Utility**	Random	**Utility**
0 (LP)	1.761	1.761	1.761	1.761	1.761
5	1.729	1.704	1.603	1.729	**1.532**
10	1.701	1.651	1.517	1.696	**1.371**
15	1.671	1.599	1.466	1.653	**1.247**
20	1.635	1.557	1.438	1.619	**1.145**
25	1.601	1.519	1.427	1.586	**1.061**
30	1.574	1.489	1.425	1.546	**0.989**
35	1.542	1.465	1.424	1.502	**0.934**
40	1.512	1.447	1.424	1.469	**0.921**
max	1.424	1.424	1.424	0.895	0.895

Table 3: Average uncertainty of MILP$_X$ for nodes of the third layer, for X with K ReLU nodes of the (1st and) 2nd layer, chosen by our **Utility** function vs Huang et al. (2020) vs random choice.

The Utility function selects very important neurons: to achieve the same accuracy than Huang et al. (2020), 2.5 time fewer nodes (10 vs 25) are necessary when picking in the same previous Layer 2. The ability from **Utility** to compare neurons from different layers enables even better frugality: 4 time fewer nodes (5 vs 20, 10 vs 40) are necessary to reach the same accuracy than Huang et al. (2020). Overall, choosing 35 neurons by **Utility** improves accuracy by 95% of what can be done if all $|X| = 116$ nodes are opened compared with LP (corresponding to $|X| = 0$).

5.2 Comparison with α, β-Crown

We conducted our evaluation on the neural networks tested in Wang et al. (2021) which display a large number ($\geq 20\%$) of images undecided by α, β-Crown. That is, these DNNs are *hard* to verify. Namely, these are 6 ReLU-DNNs: 5 MNIST DNN that can be found in the ERAN GitHub (the 4th to the 8th DNNs provided) as well as 1 CIFAR CNN from Balunovic & Vechev (2020), see also Dathathri et al. (2020), which can be downloaded from the α, β-Crown GitHub. We commit to the same ϵ settings as in Wang et al. (2021), that are recalled in Table 1. For reference, we also report an easy but very large ResNet Network for CIFAR10, already tested with α, β CROWN. We report in Table 4 the % of undecided images, that is the % of images than can be neither falsified

Network	α, β-Crown TO=10s	α, β-Crown TO=30s	α, β-Crown TO=2000s	Refined β-Crown	**Hybrid MILP**
MNIST 5×100	57% (6.9s)	55% (18.9s)	50% (1026s)	13% (92s)	13% **(46s)**
MNIST 5×200	50% (6.5s)	47% (17s)	46% (930s)	9% (80s)	**8% (71s)**
MNIST 8×100	63% (7.2s)	58% (20s)	58% (1163s)	21% (102s)	**15% (61s)**
MNIST 8×200	56% (6.8s)	55% (18s)	54% (1083s)	16% (83s)	**8% (78s)**
MNIST 6×500	53% (6.4s)	51% (16s)	50% (1002s)	–	**10% (402s)**
CIFAR CNN-B-adv	28% (4.3s)	22% (8.7s)	20% (373s)	–	**11% (417s)**
CIFAR ResNet	0% (2s)	0% (2s)	0% (2s)	–	0% (2s)

Table 4: Undecided images (%, *lower is better*) as computed by α, β-Crown, Refined β-Crown and Hybrid MILP on 7 DNNs (average runtime per image). The 6 first DNNs are hard instances. The last DNN (ResNet) is an easy instance (trained using Wong to be easy to verify, but with a very low accuracy level), provided for reference.

(by α, β-CROWN) nor verified by the tested verifier, among the 100 first images for each MNIST or CIFAR10 benchmark. The exact same DNNs and ϵ are used in Tables 1 and 4.

Analysis: on easy instances, Hybrid MILP is virtually similar to α, β-CROWN, as it is called first and it is sufficient to have 0% undecided images, as shown even on the very large ResNet.

On hard instances (the 6 first DNNs tested), Hybrid MILP is very accurate, only leaving 8%-15% of images undecided, with runtime taking less than $500s$ in average per image, and even 10 times less on smaller DNNs. It can scale up to quite large hard DNNs, such as CNN-B-Adv with 2M parameters and 20K activations.

Compared with α, β-Crown with a time-out of TO=2000s, it is much more accurate, with a reduction of undecided images by $9\% - 43\%$. It is also from 20x faster on smaller networks to similar time on the largest DNN. Compared with α, β-Crown with a time-out of TO=30s, the accuracy gap is even larger (e.g. 11% for CNN-B-Adv, i.e. half undecided images), although the average runtime is also obviously larger (solving hard istances takes longer than solving easy instances).

Last, compared with *Refined* β-Crown, we can observe three patterns: on the shallowest DNNs (5×100, 5×200), Refined β-Crown can run full MILP on almost all nodes, reaching almost the same accuracy than Hybrid MILP, but with longer runtime (up to 2 times on 5×100). On intermediate DNNs (8×100, 8×200), full MILP invoked by Refined β-Crown can only be run on a fraction of the neurons, and the accuracy is not as good as Hybrid MILP, with $6\% - 8\%$ more undecided images (that is double on 8×200), while having longer runtime. Last but not least, Refined β-Crown cannot scale to larger instances (6×500, CNN-B-Adv), while Hybrid MILP can.

5.3 Finer Grain evaluation of Accuracy

In order to evaluate the accuracy of Hybrid MILP in a finer way, showcasing the capabilities to have a very accurate and efficient verifier, we consider a quantitative questions for each image, rather than a pure yes (verified)/ no (not verified) question. Namely, we compute the ϵ for which the verifier can certify local-robustness around a given image, which makes sense as there is little rationale in setting a particular ϵ (which is however the usual local-robustness setting).

For that, we considered the challenging DNN MNIST 6×500, and the 20 first images from the MNIST benchmark. We first run the attack from α, β-Crown for varying ϵ, using a binary search, to set up an upper bound on ϵ (the initialization is $\epsilon \in [0, 1]$). Then we run binary search with a global time-out of $10000s$, initialized from 0 to this upper bound, where each call is either to α, β-Crown with TO=2000s, or Hybrid MILP. We report the results (upper bound, best bound verified by α, β-Crown and by Hybrid MILP) for each image. We report the results in Fig. 2.

Analysis: in 90% of the cases, Hybrid MILP is very close to the upper bound. On 2 images (10 and 13), Hybrid MILP is far from the upper bound: these are also the ones where the upper bound is the highest, so it is possible that the falsifier missed a closer attack to robustness.

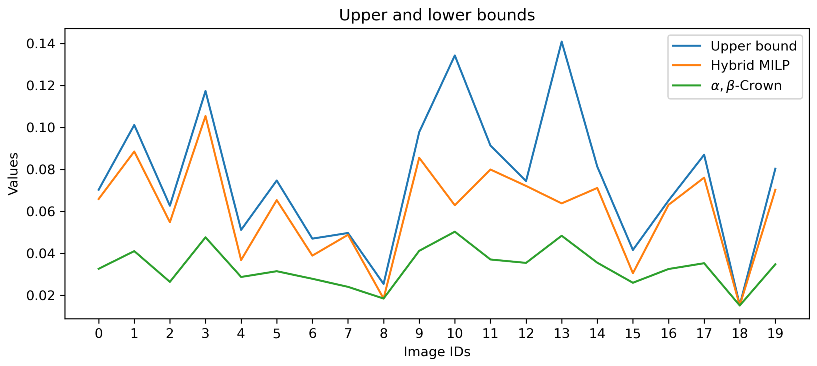

Figure 2: ϵ-robustness proved after $10000s$ for 6×500 on each of the 20 first images of MNIST.

Compared with α, β-CROWN, Hybrid MILP is much closer to the upper bound, except for 2 cases (images 8 and 18) where α, β-CROWN is already very close to the upper bound. On average, the ϵ-gap to upper bound is 0.014 for Hybrid MILP, 3 times smaller than the 0.042 gap for α, β-CROWN.

5.4 Comparison with other Verifiers?

We voluntarily limited the comparison to α, β-Crown because it is the most efficient verifier available to date, and it enabled us to consider a spectre of parameters to understand α, β-Crown scaling without too much clutter.

Notice that results for several other verifiers (PRIMA Müller et al. (2022), SDP-FO Dathathri et al. (2020), etc) were already reported on these DNNs Wang et al. (2021), with unfavorable comparison vs α, β-CROWN. Further, we reported accuracy of complete verifiers, NNenum Bak (2021), Marabou Katz et al. (2019); Wu et al. (2024), respectively 4th, 3rd of the last VNNcomp'24 Brix et al. (2024), as well as full MILP Tjeng et al. (2019) in Table 2, showing that these verifiers are not competitive on hard instances either, even on the smallest hard DNN. Concerning MnBAB Ferrari et al. (2022), we tested it in appendix, and it compares slightly unfavorably in time and accuracy towards α, β-CROWN on CNN-B-Adv and *hard* MNIST DNNs at several time-out settings. Last, Pyrat Durand et al. (2022) (2sd in the latest VNNComp) is not open source, which made running it impossible.

6 Conclusion

In this paper, we studied the LP relaxation of the exact MILP encoding of ReLU-DNNs. It allowed us to accurately assess when the relaxation of a node impacts very little the accuracy, and when it impacts the accuracy importantly, in which case representing it with a binary/integer variable is preferable. This allows us to select a small set of important neurons to encode as a binary variables, improving accuracy drastically. Our automated choice of opened neurons necessitates $>$ 4x less integer variables than previous proposals Huang et al. (2020) for the same accuracy. Our empirical studies revealed that Hybrid MILP, can yield highly accurate results, verifying up to 40% more images than the SOTA (α, β-Crown, winner of the 4 last VNNComp), with the same runtime. The reason is that α, β-Crown can scale to large easy DNNs, but does not scale to hard even small DNNs: it hits a complexity barrier, similarly as other compeating solutions. This opens a lot of perspectives, among which: verifying efficiently other hard instances; certifying ϵ-robustness of images for ϵ as large as possible; verifying global rather than local properties Wang et al. (2022).

Reproducibility Statement: We tested twice outlier results to confirm them, making sure of reproducibility on the given hardware. Precise details on the settings used are provided in the appendix. Additional results are also provided in the appendix. Tested DNNs as well as MNIST and CIFAR10 DataSet are freely available. The source code of Hybrid MILP will be provided on GitHub after acceptance (needing Gurobi as well as α, β-Crown).

REFERENCES

Stanley Bak. nnenum: Verification of relu neural networks with optimized abstraction refinement. In *NASA Formal Methods Symposium*, pp. 19–36. Springer, 2021.

Mislav Balunovic and Martin Vechev. Adversarial training and provable defenses: Bridging the gap. In *International Conference on Learning Representations (ICLR'20)*, 2020.

Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T. Johnson. The fourth international verification of neural networks competition (vnn-comp 2023): Summary and results, 2023a. URL `https://sites.google.com/view/vnn2023`.

Christopher Brix, Mark Niklas Müller, Stanley Bak, Taylor T. Johnson, and Changliu Liu. First three years of the international verification of neural networks competition (vnn-comp), 2023b.

Christopher Brix, Stanley Bak, Changliu Liu, Taylor T. Johnson, David Shriver, and Haoze (Andrew) Wu. 5th international verification of neural networks competition (vnn-comp'24), 2024. URL `https://sites.google.com/view/vnn2024`.

Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and M Pawan Kumar. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(42):1–39, 2020.

Sumanth Dathathri, Krishnamurthy Dvijotham, Alexey Kurakin, Aditi Raghunathan, Jonathan Uesato, Rudy R Bunel, Shreya Shankar, Jacob Steinhardt, Ian Goodfellow, Percy S Liang, et al. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. *Advances in Neural Information Processing Systems*, 33:5318–5331, 2020.

Serge Durand, Augustin Lemesle, Zakaria Chihani, Caterina Urban, and Francois Terrier. Reciph: Relational coefficients for input partitioning heuristic. In *1st Workshop on Formal Verification of Machine Learning (WFVML 2022)*, 2022. URL `https://pyrat-analyzer.com/`.

Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15*, pp. 269–286. Springer, 2017.

Claudio Ferrari, Mark Niklas Mueller, Nikola Jovanović, and Martin Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. In *International Conference on Learning Representations (ICLR'22)*, 2022.

Chao Huang, Jiameng Fan, Xin Chen, Wenchao Li, and Qi Zhu. 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. doi: 10.1109/TCAD.2020.3013071.

Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In Rupak Majumdar and Viktor Kunčak (eds.), *Computer Aided Verification*, pp. 97–117, Cham, 2017. ISBN 978-3-319-63387-9.

Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David L. Dill, Mykel J. Kochenderfer, and Clark Barrett. The marabou framework for verification and analysis of deep neural networks. In Isil Dillig and Serdar Tasiran (eds.), *Computer Aided Verification*, pp. 443–452, Cham, 2019. Springer International Publishing. ISBN 978-3-030-25540-4.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *International Conference on Learning Representations (ICLR'18)*, 2018.

Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pp. 3578–3586. PMLR, 2018.

Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. Prima: General and precise neural network certification via scalable convex hull approximations. *Proc. ACM Program. Lang.*, 6(POPL), jan 2022. doi: 10.1145/3498704.

Mark Niklas Müller, Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T. Johnson. The third international verification of neural networks competition (vnn-comp 2022): Summary and results, 2022. URL `https://sites.google.com/view/vnn2022`.

Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. Beyond the single neuron convex barrier for neural network certification. *Advances in Neural Information Processing Systems*, 32, 2019a.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL), jan 2019b. doi: 10.1145/3290354.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. Robustness certification with refinement. In *International Conference on Learning Representations (ICLR'19)*, 2019c.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *International Conference on Learning Representations (ICLR'14)*, 2014.

Christian Tjandraatmadja, Ross Anderson, Joey Huchette, Will Ma, Krunal Kishor Patel, and Juan Pablo Vielma. The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. *Advances in Neural Information Processing Systems*, 33:21675–21686, 2020.

Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *International Conference on Learning Representations (ICLR'19)*, 2019.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 29909–29921. Curran Associates, Inc., 2021.

Zhilu Wang, Chao Huang, and Qi Zhu. 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.

Haoze Wu, Omri Isac, Aleksandar Zeljić, Teruhiro Tagomori, Matthew Daggitt, Wen Kokke, Idan Refaeli, Guy Amir, Kyle Julian, Shahaf Bassan, Pei Huang, Ori Lahav, Min Wu, Min Zhang, Ekaterina Komendantskaya, Guy Katz, and Clark Barrett. Marabou 2.0: A versatile formal analyzer of neural networks. In Arie Gurfinkel and Vijay Ganesh (eds.), *Proceedings of the 36th International Conferenceon Computer Aided Verification (CAV '24)*, volume 14681 of *Lecture Notes in Computer Science*, pp. 249–264. Springer, July 2024. Montreal, Canada.

Dong Xu, Nusrat Jahan Mozumder, Hai Duong, and Matthew B. Dwyer. Training for verification: Increasing neuron stability to scale dnn verification. In Bernd Finkbeiner and Laura Kovács (eds.), *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 24–44, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-57256-2.

Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. 2021.

Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. General cutting planes for bound-propagation-based neural network verification. In *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.

APPENDIX

SETTING FOR HYBRID MILP

Hybird MILP first call α, β-Crown with short time-out (TO), then call partial MILP on those inputs which was neither certified nor falsified by this run of α, β-Crown. We are using two settings of TO, for smaller DNNs we use T0= $10s$, and for the two larger ones, we use TO= $30s$.

The setting for partial MILP for fully-connected DNNs is about how many neurons need to be opened (once set, the selection is automatic). The runtime depending crucially upon the number of open ReLU neurons, we set it quite tightly, only allowing few neuron deviation to accommodate to a particularly accurate/inaccurate bound computation (measure by the weight of the remaining Utility function). As complexity increases with the layer considered, as the size of the MILP model grows, we lower this number with the depth, only committing to an intermediate number for the output neuron (the number of output neurons is smaller than hidden layer, and this is the most important computation). We experimentally set this number so that each computing the bounds in each hidden layer takes around the same time. Remember that in layer 1, partial MILP is not necessary and propagating bounds using interval arithmetic is already exact. We open [48,48] to compute bounds for hidden layer 2, [21,24] for layer 3, [11,14] for layer 4, [6,9] for layer 5, [3,6] for layer 6, [2,5] for layer 7, [1,4] for hidden layer 8 (if any), and we open [14,17] for the output layer.

For convolutional CNNs, the strategy is adapted, as there is much more neurons, but in a shallower architecture and not fully connected. The second layer is computed accurately, opening 200 neurons, which is manageable as there is only one ReLU layer to consider, and accuracy here is crucial. We do not open any nodes in the third layer (the first fully connected layer) if the output layer is the next one (which is the case for CNN-B-Adv), and instead rely on the choice of important nodes for the output layer. Otherwise, we open 20 neurons. In the output layer, we open at least 45 neurons (there is less output neurons than nodes in the previous layer), and enlarge the number of open neurons (up to 300) till we find an upper bound, that is a best current MILP solution, of around +0.1 (this 0.1 was experimentally set as target, a good balance between accuracy and efficiency), and compute a guaranteed lower bound (the goal is to guarantee the bound is > 0).

In Table 5, we sum-up the TO and minimum open numbers for each DNN considered.

Network	TO for α, β-Crown	Minimum number of Open neurons
MNIST 5×100	10s	48,21,11,6,14
MNIST 5×200	10s	48,21,11,6,14
MNIST 8×100	10s	48,21,11,6,3,2,1,14
MNIST 8×200	10s	48,21,11,6,3,2,1,14
MNIST 6×500	30s	48,21,11,6,3,14
CIFAR CNN-B-adv	30s	200, 0, 45

Table 5: Settings of Hybrid MILP for the different *hard* instances

α, β-Crown uses massively parallel (> 4096 threads) GPU, while Partial MILP uses 20 CPU-threads.

Notice that a different balance between accuracy and runtime could be set. For instance, we set up the numbers of open neurons to have similar runtime as Refined β-Crown for the first 4 DNNs ($50s - 100s$). We could easily target better accuracy (e.g. for 8×100 with a relatively high 15% undecided images) by increasing the number of open neurons, with a trade-off on runtime (current runtime is at $61s$). By comparison, the sweet spot for α, β-Crown seems to be around TO= $30s$, enlarging the time-out having very little impact on accuracy but large impact on runtime (see Table 1).

In the following, we provide results comparing α, β-Crown to other verifiers, to justify our use of α, β-Crown as state of the art for efficient verifiers as main source of comparison to Hybrid MILP for hard DNN instance.

Comparison α, β-Crown vs PRIMA

PRIMA Müller et al. (2022) is a major verifier in the ERAN toolkit. In Table 6, we report the comparison between PRIMA and α, β-Crown, mainly from Wang et al. (2021). The setting is mainly similar from ours, but numbers are not perfectly comparable as the images tested are not exactly the same (1000 first or 200 first images for CNN-B-Adv), vs 100 first in Tables 4, 1. Also, time-out settings and hardware are slightly different. The overall picture is anyway the same.

Network	α, β-Crown	Refined β-Crown	PRIMA
MNIST 5×100	N/A	14.3% (102s)	33.2% (159s)
MNIST 5×200	N/A	13.7% (86s)	21.1% (224s)
MNIST 8×100	N/A	20.0% (103s)	39.2% (301s)
MNIST 8×200	N/A	17.6% (95s)	28.7% (395s)
MNIST 6×500	51% (16s)	–	64% (117s)
CIFAR CNN-B-adv	18.5% (32s)	–	27% (344s)
CIFAR ResNet	0% (2s)	–	0% (2s)

Table 6: Undecided images (%, *lower is better*), as computed by α, β-Crown, Refined β-Crown, and PRIMA, as reported in Wang et al. (2021), except for 6×500 that we run ourselves. N/A means that Wang et al. (2021) did not report the numbers, while $-$ means that Refined β-Crown cannot be run on these DNNs.

Analysis: On the 4 smallest MNIST networks, PRIMA uses a refined path comparable with Refined β-Crown. However, it is slower and less accurate than Refined β-Crown. On larger *hard* networks, PRIMA has also more undecided images than α, β-Crown, while the runtime is > 5 times larger. Hence, Hybrid MILP is more accurate than Prima with similar runtime or faster.

Notice that kPoly Singh et al. (2019a), OptC2V Tjandraatmadja et al. (2020), SDP-FO Dathathri et al. (2020) numbers were also reported in Wang et al. (2021) on these networks, with even more unfavorable results.

Comparison α, β-Crown vs MN-BaB

MN-BaB Ferrari et al. (2022) is an improvement built over PRIMA, using a similar Branch and Bound technique as used in α, β-Crown. Results in Ferrari et al. (2022) are close to those of α, β-Crown. However, none of the *hard* networks from Wang et al. (2021) that we consider have been tested. We thus tested three representative *hard* DNNs (first 100 images) to understand how MN-BaB fairs on such hard instances, and report the numbers in Table 7. Results are directly comparable with Table 4.

	α, β-Crown	α, β-Crown	MN-BaB	MN-BaB
Network	TO=30s	TO=2000s	TO=30s	TO=2000s
MNIST 5×100	55% (19s)	50%(1026s)	60% (19s)	50% (1027s)
MNIST 6×500	51% (16s)	50% (1002s)	58% (18s)	55% (1036s)
CIFAR CNN-B-adv	22% (8.7s)	20% (373s)	43% (14s)	24% (576s)

Table 7: Undecided images (%, *lower is better*), as computed by α, β-Crown, and MN-BaB

Analysis: results reveal that MN-BaB is slightly slower and slightly less accurate than α, β-Crown. Notice the specially high number of undecided images for CNN-B-Adv with TO=30s, probably meaning that 30s is too small for MN-BaB on this large DNN. Hence, Hybrid MILP is more accurate than MN-BaB with similar runtime or faster.

Comparison α, β-Crown vs NNenum

NNenum Bak (2021) is a complete verifier with good performance according to VNNcomp. It was the only complete verifier tested in Table 2 to verify more images than α, β-Crown. The experiments section in Bak (2021) does not report the *hard* DNNs we are considering. We tried to experiment it on the same MNIST 6×500 and CIFAR CNN-B-adv as we did in Table 7 for MN-BaB. Unfortunately, on 6×500, buffer overflow were reported. We report in Table 8 experiments with the same 2000s Time-out (it was $10000s$ in Table 2) for a fair comparison with α, β-Crown, on both MNIST 5×100 and CIFAR CNN-B-Adv. On MNIST 5×100, NNenum is slightly more accurate than α, β-Crown, but far from the accuracy Hybrid MILP. On CIFAR CNN-B-adv, NNenum was much less accurate than α, β-CROWN, and thus of Hybrid MILP. In both test, the runtime of NNenum was also much longer than for Hybrid MILP.

Network	α, β-Crown TO=2000s	NNenum TO=2000s	Hybrid MILP
MNIST 5×100	50%(1026s)	44% (1046s)	**13% (46s)**
CIFAR CNN-B-adv	20% (373s)	40% (1020s)	**11% (417s)**

Table 8: Undecided images (%, *lower is better*), as computed by α, β-Crown and NNenum with 2000s time-out, and Hybrid MILP

.

Ablation studies

In this section, we consider ablation studies to understand how each feature enables the efficiency of pMILP.

Time scaling with open nodes

First, we explore the time scaling with different number of open nodes, for our full Utility function using nodes in the last two layers (Layer 1 and 2), providing finer details than in Table 3, with the same setting, i.e. previous layer being computed with full MILP.

$\lvert X\rvert$	Time	Uncertainty
0	2.6	1.760946128
1	7.3	1.702986873
2	11.1	1.65469034
3	16.3	1.612137282
4	15.5	1.571001109
5	15.7	1.531925404
6	15.8	1.49535638
7	16.4	1.46189314
8	15.8	1.4299535
9	17.2	1.4006364
10	22.5	1.3711203
11	27.2	1.3438245
12	21.6	1.3183356
13	28.7	1.2938690
14	29.6	1.2690507
15	24.5	1.2475106

$\lvert X\rvert$	Time	Uncertainty
16	31.9	1.2243065
17	28.6	1.2031791
18	30.4	1.1839474
19	34.0	1.1644653
20	42.1	1.1456181
21	47.6	1.1261252
22	62.7	1.1089745
23	70.0	1.0931242
24	70.8	1.0773088
25	139.9	1.060928
26	154.2	1.045715
27	213.1	1.030605
28	211.3	1.016058
29	373.1	1.001374
max=116	3300	0.895

Table 9: Time and uncertainty scaling of pMILP with number of nodes.

The exponential complexity with the number of nodes is obvious, as shown on Figure 3, where time is represented using logarithmic scale. Notice that when certifying, pMILP uses $|X| \in$ 21-24, which is is a good trade off between time and accuracy.

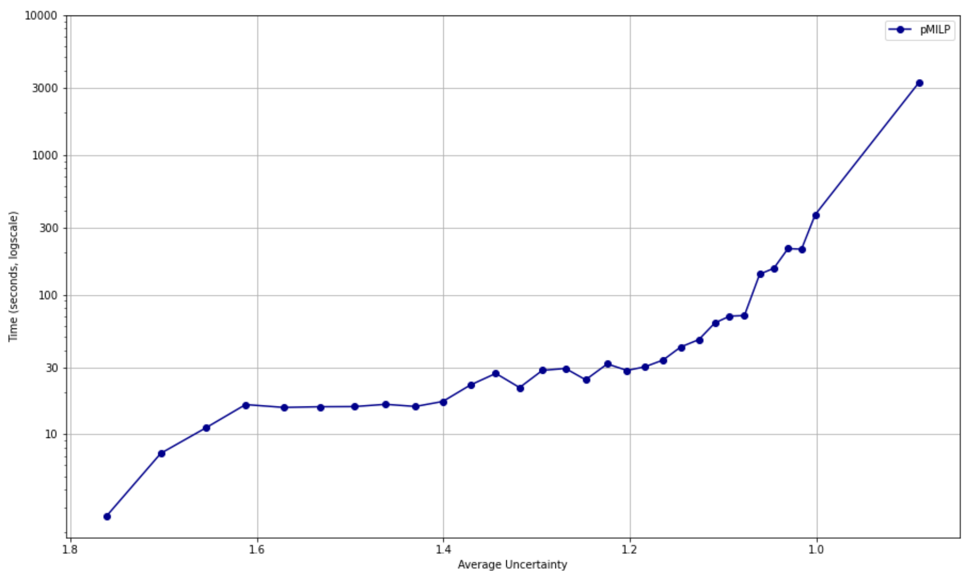

Figure 3: Time and uncertainty scaling of pMILP with number of nodes. Time is using logscale.

USEFULNESS OF COMPUTING PREVIOUS LAYERS ACCURATELY

Then, we explore the usefulness of computing accurately each layer inductively. For that, we keep the setting of Table 3 and Table 9, but computing the previous layer with LP rather than with full MILP.

$\|X\|$	Time	Uncertainty with LP for layer 2 (resp. MILP for layer 2)
5	9.3	3.24737 (1.532)
10	10.6	3.02214 (1.371)
15	11.9	2.82383 (1.247)
20	13.1	2.63862 (1.145)
25	16.0	2.47324 (1.061)
30	28.3	2.32793 (0.989)
35	48.1	2.19506 (0.934))
40	89.4	2.07107 (0.921)

Table 10: Comparison of accuracy in layer 3 from computing layer 2 inaccurately using LP vs (from computing layer 2 accurately using MILP)

This experiment explains the rationale to use divide and conquer protocol, using many calls (one for each neuron) with relatively small number $|X|$ of open nodes rather than fewer call to MILP with larger number $|X|$ of open nodes. This is clear already with only 1 layer before.

RESTRICTING OPEN NODES VS TIME-OUTS

Running full MILP till a small MIP-Gap (typically 0.001) is reached is extremely time inefficient.

Instead, the standard strategy is to set a reasonable time-out and use whatever bound has been generated. We compare this standard strategy with the pMILP strategy of setting a priori a number of open nodes.

$\lvert X\rvert$	Time	Uncertainty
1	14	3.233021901
2	15.2	3.140309921
3	17.21	3.059083103
4	17.4	2.986166762
5	19.2	2.856229765
6	20.9	2.799248232
7	23.7	2.746167245
8	26.6	2.69485246

(a) pMILP

Time	Uncertainty
21.1	3.348236261
27.6	3.24604282
38.2	3.196640184
47.1	3.164298172
56.7	3.146913614
106.7	3.108035223
156.3	2.900438725
205.8	2.848648426
406.7	2.800268264
606.1	2.737064255

(b) full MILP

Table 11: Comparison of bounding the number of nodes for pMILP and using different time outs for full MILP. In both settings, lower and upper bounds of previous layers are the same (computed by pMILP).

pMILP obtains 2.8 accuracy in $<$ 21 seconds (with 7 open nodes), while full MILP needs 400 seconds to obtain it, a 19x speed up. For 2.7 accuracy, the speedup is $>>$ 22.

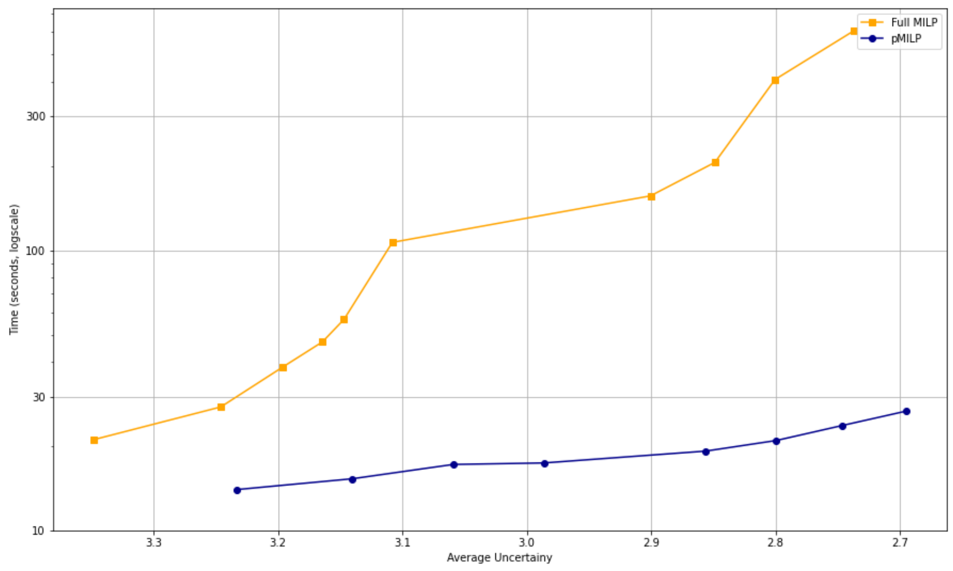

Figure 4: Comparison of uncertainty at layer 7 for full MILP with different time-outs vs pMILP with different number of open nodes. Time is using logscale.

Figure 4 shows that choosing nodes is much more efficient for time/accuracy trade-off than setting time outs and use full MILP. And this is for the smallest DNN we considered (500 hidden neurons, far from the biggest 20k neuron DNN we experimented with)

Formula for Utility

Recall that for nodes a, b, we use $\hat{a}, \hat{b}$ to denote variable after ReLU functions, and $\Delta(a), \Delta(\hat{a}), \Delta(b), \Delta(\hat{b})$ to denote the changes of those variables. a means the source node, and b means nodes one layer after a, and z is 2 layers after a and one after b.

The utility function for a node a wrt neuron z is defined inductively as follows:

$$\begin{aligned}
\Delta(\hat{a}) &= \text{ReLU}(\text{sol}(a)) - \text{sol}(\hat{a}) \\
\Delta(b) &= W_{ab}\Delta(\hat{a}) \\
\Delta(\hat{b}) &= \begin{cases} \frac{\text{UB}(b)}{\text{UB}(b)-\text{LB}(b)}\Delta(b), & \text{if } W_{bz} > 0 \\ \max(\Delta(b), -\text{sol}(b)), & \text{if } W_{bz} < 0 \text{ and } \text{sol}(b) \geq 0 \\ \max(\Delta(b) + \text{sol}(b), 0), & \text{if } W_{bz} < 0 \text{ and } \text{sol}(b) < 0 \end{cases} \\
\text{Utility\_max}^z(a) &= -\sum_{b \in \ell} W_{bz}\Delta(\hat{b})
\end{aligned}$$