Generation and Mitigation of Precision-Based Evasion Attacks in NNs Verification

Andrea Gimelli, Stefano Demarchi, Davide Anguita, Armando Tacchella, and Luca Oneto
University of Genoa - Via Opera Pia 11a, 16145, Genova, Italy

***Abstract*—Adversarial examples exploit small perturbations in the input data to manipulate the decisions of neural network classifiers. To address this challenge, verification tools have been developed to certify a range of perturbations that cannot alter the prediction of a classifier. Nevertheless, classifiers can be implemented using different precisions and arithmetic due to computational or hardware constraints. Recent studies [21], [54] reveal that discrepancies between the implemented and the verified classifier can be exploited to invalidate existing guarantees, since verifiers predominantly rely on floating-point arithmetic. In this work, we show that state-of-the-art verifiers (i.e., the podium of the VNN-COMP) remain vulnerable to this class of attacks by introducing a new, computationally efficient, precision-based evasion attack. We then investigate how augmenting a verifier with interval-based arithmetic can effectively mitigate these attacks, thereby enabling precision-aware verification. As our empirical evaluation confirms, our approach ensures consistent guarantees and successful mitigation.**

***Index Terms*—Neural Network Verification, Evasion Attacks, Precision-Based Attacks, Precision-Aware Verification, Interval-Based Arithmetic.**

I. Introduction

Formal verification is being popularized as a tool for Machine Learning (ML) practitioners in order to provide mathematical guarantees on the behavior of their models. In particular, verification of Neural Networks (NNs) [9]–[11], [14]–[16], [22], [25], [35], [36], [40], [42], [45], [46] has gained considerable momentum in recent years, following the discovery of robustness vulnerabilities [1], [12], [41]. Indeed, NNs are sensitive to *adversarial attacks* - small perturbations to input data that can manipulate the decision of a classifier - often resulting in incorrect outputs. Such failures pose a serious threat in safety- or security-critical domains.

In this work we approach a threat to NN verification that leverages the fact that, in practice, many models are deployed using a different precision and/or arithmetic from the original trained model [7], [29], [33], [47]. This is usually necessary (i) to balance computational requirement and model accuracy and/or (ii) to cope with limitations of the hardware on which the model has to run. The concerning aspect is that NN verifiers are not currently able to take this information into account: although some of them model the accumulation of floating-point arithmetic error in their structure [22], [27] they do not consider the precision of the implementation they verify. It is important to highlight that it is not enough to just represent the model parameters using the same precision of the implementation, because it is the actual arithmetic that will change the behavior of a model. Finally, it is also possible that the implementation precision could be mixed, partially unknown or even not available, e.g., using custom-made hardware [13], [17], [24], [37].

To the best of our knowledge, this issue has only been partially investigated by [54] and [21], who demonstrated how rounding errors and differing implementations of a verified NN can mask an adversarial attack. However, their methodologies rely on overly complex and computationally expensive procedures, or they demand the attacker to possess more advanced capabilities than merely altering the precision or arithmetic. In fact, they either do not leverage the extensive body of research on adversarial examples [2], [34] or they require multiple calls to NN verifiers, making them impractical in real-world applications. In particular, [21] proposes an algorithm to generate adversarial examples within a NN's provably robust region. However, this approach requires iterative verification of robustness specifications while searching for a vulnerable point. Meanwhile, in [54], the attacker can modify not only the precision of the NN's arithmetic but also its architecture. Finally, although these works suggest possible avenues to address such attacks, they do not provide a fully developed mitigation strategy.

Starting from the ideas of [21], in this work we propose a new attack strategy that requires minimal attacker capabilities – specifically, the ability to modify only the arithmetic of the NN implementation, without altering its architecture or parameter values, and without any need to invoke verification tools – while leveraging the extensive results in adversarial ML [2], [34]. We show how our method applies to linear binary and multiclass classifiers, as well as to non-linear Rectified Linear Unit (ReLU) based NNs (both binary and multiclass). Next, we demonstrate that our attacks can successfully deceive the three top-ranked verifiers from the last two years of the International Competition of Neural Network Verification (VNN-COMP) [5], [6], namely α,β-CROWN [45], [49]–[51], PyRAT [11], [27], and Marabou [23], [46], as well as the runner-up NeVer2 [10], since its implementation is more readily accessible and more easily modifiable than the other tools. We then propose and implement a mitigation strategy that employs interval arithmetic for both inputs and model parameters, enhancing NeVer2 with the ability to detect potential vulnerabilities arising from a given NN arithmetic implementation. We show that this approach can successfully detect precision-based attacks and provide a precision-aware verification strategy, thereby guaranteeing the robustness of a NN under a specified implementation precision. We have made all the code used to reproduce these experiments publicly available[1].

The remainder of the paper is structured as follows. Section II provides definitions and context on NN verification. Section III details the threat model considered in this study.

[1]https://github.com/NeVerTools/IJCNN-2026

Section IV presents the newly proposed precision-based attacks on NN verification, demonstrating their performance on real-world examples. Section V introduces our mitigation strategy, supported by experimental results that show its effectiveness in detecting the attacks described in Section IV. Finally, Section VI concludes the paper by discussing our findings and outlining future research directions.

II. Verification of Neural Networks

Given an input space $\mathcal{X}$ and an output space $\mathcal{Y}$, a NN is a function $f : \mathcal{X} \to \mathcal{Y}$ that maps inputs to outputs. Formal verification aims to algorithmically prove that a certain *specification* holds, i.e., that the NN satisfies certain *post-conditions* on its output, assuming specific *pre-conditions* on its input. More formally, given two bounded sets $I \subset \mathcal{X}$ and $O \subset \mathcal{Y}$, we seek to prove that

$$\mathbf{x} \in I \Rightarrow f(\mathbf{x}) \in O. \tag{1}$$

Because proving Property (1) is generally challenging [38], verification tools typically seek a *counterexample*, i.e., a solution to

$$\exists \mathbf{x} \in I : f(\mathbf{x}) \notin O. \tag{2}$$

Although this query cannot express certain properties of NNs, such as invertibility or equivalence [28], it captures the general problem of testing resilience to adversarial examples.

In this work, we focus on the verification tools that took the top spots in the latest VNN-COMP [5], namely α,β-CROWN, PyRAT, and Marabou, as well as the runner-up NeVer2 which we consider as a basis for our improvements to the verification strategy. Although there exist many other verifiers that do not participate in the competition, these four are among the most popular and therefore more likely to be employed in real-world case studies.

α,β-CROWN is an efficient NN verifier built on the linear bound propagation framework auto-LiRPA [49] and incorporates a series of previous works on bound-propagation-based NN verifiers: CROWN [51], α-CROWN [50], β-CROWN [45], and GCP-CROWN [51]. Its core techniques combine the efficient, GPU-accelerated linear bound propagation method with branch-and-bound techniques specialized for NN verification.

PyRAT [11], [27] (Python Reachability Assessment Tool) is a verification framework based on abstract interpretation. Thanks to its approach specifically tailored to handle the high dimensionality of NNs, PyRAT efficiently applies abstract interpretation while leveraging tensor operations. It supports a wide range of neural architectures and layers, from small tabular networks to more complex designs with convolutional layers and skip connections.

Marabou [23], [46] is a user-friendly NN verification tool that formulates network properties as constraint satisfaction problems. It includes Python and C++ APIs, allowing users to load a NN and define arbitrary linear properties over it.

NeVer2 [10] is an open-source, cross-platform tool for designing, training, and verifying NNs. It uses the pyNeVer [16] Python API for verification, which employs an abstraction-refinement algorithm. This algorithm uses symbolic bounds propagation to identify stable and unstable neurons, combined with iterative refinement to construct a search tree that proves either the safety or the unsafety of the specified verification query.

III. Threat Model

In this section, we introduce the threat model addressed in this work, namely a systematic description of how an adversary could attack a verifier for an ML-based classifier [20], [28], [48]. Specifically, this model describes the adversary's capabilities, knowledge, and goals, as well as the points in the system where an attack may be mounted.

Consider an *ideal classifier* [3], [39], denoted by $f_\theta : \mathbb{R}^d \to \{1, \dots, c\}$, where θ represents the parameters defining the classifier[2]. For an input $\mathbf{x} \in \mathbb{R}^d$, the classifier outputs $y = f(\mathbf{x})$ with $y \in \{1, \dots, c\}$. A *floating-point representation* of f, denoted by $\tilde{f}$, approximates f using floating-point arithmetic. In this case, $\tilde{\mathbf{x}}$ and $\tilde{\theta}$ are the floating-point representations of $\mathbf{x}$ and θ, respectively. Verification using state-of-the-art solvers applies directly to $\tilde{f}$ (rather than f). A *real implementation* [7], [29], [33], [47] of $\tilde{f}$, denoted by $\tilde{f}^p$, may be necessary to reduce computational overhead or because the target hardware supports only a specific (standard or custom) arithmetic [13], [17], [24], [37]. Under $\tilde{f}^p$, the floating-point values $\tilde{\mathbf{x}}$ and $\tilde{\theta}$ change representation into $\tilde{\mathbf{x}}^p$ and $\tilde{\theta}^p$, respectively. In general, $\tilde{\mathbf{x}}^p$, $\tilde{\theta}^p$, and the arithmetic used in $\tilde{f}^p$ might each involve different or unknown precision levels [53].

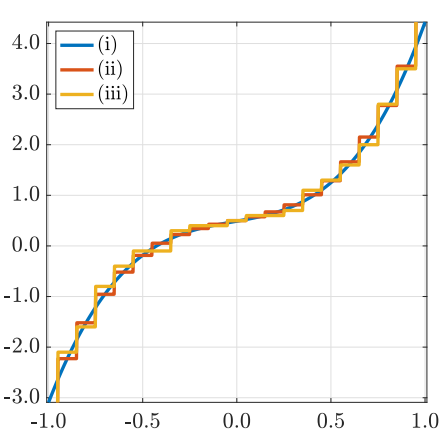
The attacker's objective is to exploit the fact that the verifier checks $\tilde{f}$ rather than $\tilde{f}^p$. We assume the verifier knows precisely how $\tilde{\mathbf{x}}^p$ and $\tilde{\theta}^p$ are produced (fixed-point precision with p fractional digits), but does *not* know the exact arithmetic used by $\tilde{f}^p$ (i.e., how computations are performed at fixed-point precision p). Moreover, note that current best state-of-the-art solvers can only handle standard floating-point arithmetic during verification [5], [6].

This scenario is both more realistic and more challenging for the attacker compared to one where the attacker is aware only of $\tilde{f}$. In that simpler setting, an attacker could manipulate both the precision of $\tilde{\mathbf{x}}^p$ and $\tilde{\theta}^p$ *and* the arithmetic of $\tilde{f}^p$. Here, by contrast, the attacker may only alter the arithmetic within $\tilde{f}^p$. For instance, in Figure 1a, consider computing a binary classifier (i) in double-precision floating-point arithmetic (`float64`) with `float64` inputs and parameters, (ii) in `float64` arithmetic with inputs and parameters at fixed-point precision p, and (iii) in p fixed-point arithmetic with inputs and parameters at fixed-point precision p. To simulate (iii), we employ `float64` arithmetic but round every variable, before and after each operation, to $p = 2$ fixed-point arithmetic.

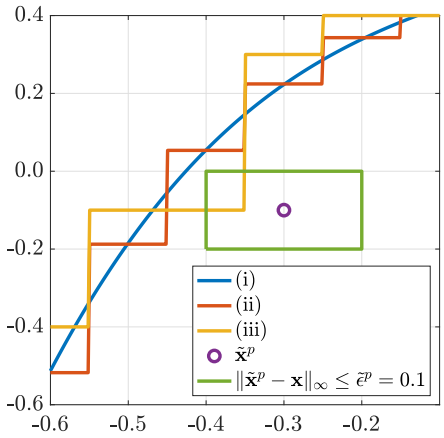
The attacker's aim is to make the verifier prove a property about $\tilde{f}$ that does not hold under $\tilde{f}^p$. Because the verifier can only check properties of $\tilde{f}$ (albeit with accurate knowledge of $\tilde{\mathbf{x}}^p$ and $\tilde{\theta}^p$), the attacker capitalizes on discrepancies between the arithmetic in $\tilde{f}$ and $\tilde{f}^p$. Concretely, the property of interest (which the attacker intends to invalidate) states that the classification of $\tilde{\mathbf{x}}^p$ remains invariant under any perturbation within $\tilde{\epsilon}^p$ (a possible value in p fixed-point arithmetic) in the ℓ_∞ norm, meaning it is not the case that

$$\exists \mathbf{x} \in \|\tilde{\mathbf{x}}^p - \mathbf{x}\|_\infty \leq \tilde{\epsilon}^p : \tilde{f}^p(\mathbf{x}) \neq \tilde{f}^p(\tilde{\mathbf{x}}^p), \tag{3}$$

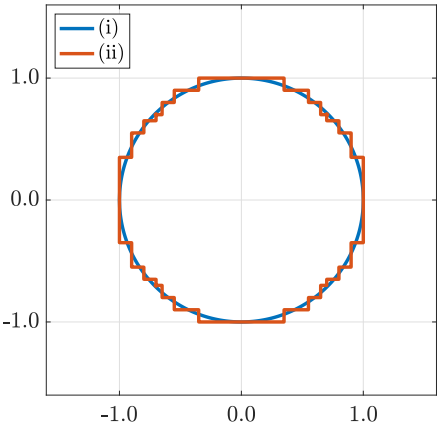
[2]From now on, we drop the subscript $\boldsymbol{\theta}$ on f to avoid cluttering the notation.



(a) Binary classifier computed: (i) in double-precision floating-point arithmetic (`float64`) with `float64` inputs and parameters, (ii) in `float64` arithmetic with inputs and parameters at fixed-point precision p=1, and (iii) in p=1 fixed-point arithmetic with inputs and parameters at fixed-point precision p=1.



(b) An example illustrating that Property (4) holds, while Property (3) does not, for the binary classifiers depicted in Figure 1a.



(c) Euclidean norm, specifically $\|\cdot\|_2 = 1$, computed (i) in double-precision floating-point arithmetic (`float64`) and (ii) in fixed-point arithmetic with p=1.

Fig. 1: Effect of a real implementation.

but what it can be actually checked with current verifiers is that is not the case that

$$\exists \mathbf{x} \in \|\tilde{\mathbf{x}}^p - \mathbf{x}\|_\infty \leq \tilde{\epsilon}^p : \tilde{f}(\mathbf{x}) \neq \tilde{f}(\tilde{\mathbf{x}}^p). \tag{4}$$

For instance, Figure 1b provides an example verifying Property (4), whereas Property (3) is not satisfied by the binary classifier shown in Figure 1a.

Adopting the ℓ_∞ norm and an implementation-dependent radius $\tilde{\epsilon}^p$ is both more challenging and realistic than other alternatives for the attacker. For example, an Euclidean norm with a free ϵ might be affected by low-level implementation details and allow an attacker to exploit the verifier's incomplete information. By contrast, the ℓ_∞ norm is robust to variations introduced by p fixed-point arithmetic, and $\tilde{\epsilon}^p$ is precisely defined and consistent with that arithmetic. Figure 1c illustrates this by comparing the computation of the Euclidean norm $\|\cdot\|_2 = 1$ (i) in double-precision floating-point arithmetic (`float64`) and (ii) in fixed-point arithmetic with $p = 1$.

IV. Precision-based attacks

In this section, we present a methodology for finding an $\tilde{\mathbf{x}}^p$ and $\tilde{\epsilon}^p$ that satisfy Property (4) but not Property (3).

Note that, in general, attacking an ML-based classifier is NP-hard [30], as is the verification problem [22]. Nevertheless, the role of an attacker is much simpler than that of a defender (or verifier), since the attacker needs to identify only one vulnerability, while the defender (or verifier) must address all possible attacks.

Some previous works [21], [54] proposed attacks exploiting representation precision, but these require solving NP-hard problems (e.g., they rely on a verifier to construct the attack). In contrast, we show that it is possible to find a successful attack in polynomial time (if the attack can succeed).

We begin with the simple case of linear classifiers in Section IV-A, where both attacking and verification can be performed in polynomial time. Under the threat model of Section III, we demonstrate that it is already possible to discover vulnerabilities, i.e., an $\tilde{\mathbf{x}}^p$ and $\tilde{\epsilon}^p$ that satisfy Property (4) but not Property (3).

Subsequently, we show how to extend the attack to non-linear classifiers in Section IV-B, focusing on shallow ReLU-based models [3], where both attacks and verification are NP-hard. This further illustrates that the proposed attack is also effective in more complex settings.

A. Linear Classification

Consider the binary classification problem with a linear classifier [39]. Let the input space be $\mathcal{X} = \mathbb{R}^d$ and the output space be $\mathcal{Y} = \{\pm 1\}$. Our model $f : \mathcal{X} \to \mathcal{Y}$ can then be written as

$$\hat{y} = f(\mathbf{x}) = \text{sign}(\mho(\mathbf{x})) = \text{sign}(\mathbf{w}'\mathbf{x}), \tag{5}$$

where $\mathbf{x} \in \mathbb{R}^d$, $\hat{y} \in \mathcal{Y}$, $\mho : \mathcal{X} \to \mathbb{R}$ and $\mathbf{w} \in \mathbb{R}^d$ are the learnable parameters. To learn $\mathbf{w}$, we make use of a training set $\mathcal{L} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$ and employ the Ridge Regression learning algorithm [31]. For this purpose, let us define

$$X = [\mathbf{x}_1', \cdots, \mathbf{x}_n'], \quad \mathbf{y} = [y_1, \cdots, y_n]', \tag{6}$$

and $I \in \mathbb{R}^{d \times d}$ as the identity matrix. Ridge Regression learning phase can be expressed as

$$\mathbf{w}^* = \arg\min_{\mathbf{w} \in \mathbb{R}^d} \|X\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 = (X'X + \lambda I)^{-1} X'\mathbf{y},$$
$$\mathbf{w}^* = {}^{\mathbf{w}^*}/_{\|\mathbf{w}\|_2}. \tag{7}$$

Ridge Regression imposes an L_2 penalty, i.e., $\|\mathbf{w}\|_2^2$, on the standard mean-squared error cost function, shrinking coefficient estimates toward zero. This penalty helps control model complexity by reducing overfitting and variance. It is particularly beneficial when predictors are highly correlated,

as it stabilizes the estimates. $\lambda \in (0, \infty)$ is the regularization hyperparameter that balances over- and under-fitting. The parameter λ can be tuned via resampling techniques [32], while the final performance of the model must be evaluated on a fresh, previously unseen dataset, the test set $\mathcal{T} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_t, y_t)\}$ [32].

Note that the solution of Problem (7) can only be obtained in practice through iterative numerical methods. Accordingly, in this setting we set $\tilde{\theta} = \tilde{\mathbf{w}}^*$, namely the `float64` solution of Problem (7). Then, we will also express it in its fixed-point representation with respect to p, denoted as $\tilde{\boldsymbol{\theta}}^p = \tilde{\mathbf{w}}^{*,p}$.

In order to find $\tilde{\mathbf{x}}^p$ and $\tilde{\epsilon}^p$ that satisfy Property (4) but not Property (3), we propose the procedure outlined in Algorithm 1. We begin with a point $(\tilde{\mathbf{z}}^p, y)$ in p fixed-point representation, drawn from a sample set $\mathcal{S} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$. Note that $\mathcal{S}$ is arbitrary, in the sense that it may or may not overlap with the sets $\mathcal{L}$ or $\mathcal{T}$. We then restrict our attention to samples correctly classified by both $\tilde{f}$ (the `float64` implementation of f) and $\tilde{f}^p$ (the p fixed-point implementation of $\tilde{f}$) with inputs and parameters at fixed-point precision p:

$$y = \tilde{f}(\tilde{\mathbf{z}}^p) = \tilde{f}^p(\tilde{\mathbf{z}}^p). \tag{8}$$

Next, we progressively move $\tilde{\mathbf{z}}^p$ toward the decision boundary (i.e., $\tilde{\mathbf{z}}^p + \tilde{\delta}^p$ at precision p) until we identify

$$y = \tilde{f}(\tilde{\mathbf{z}}^p + \tilde{\delta}^{*,p}) \neq \tilde{f}^p(\tilde{\mathbf{z}}^p + \tilde{\delta}^{*,p}), \tag{9}$$

where $\tilde{\delta}^{*,p}$ is the shift at which the two implementations disagree. If no such $\tilde{\delta}^{*,p}$ exists for the chosen $\tilde{\mathbf{z}}^p$, we select a different $\tilde{\mathbf{z}}^p$ and repeat the procedure. Once $\tilde{\delta}^{*,p}$ is found, we define $\tilde{\mathbf{x}}^p$ by moving $\tilde{\mathbf{z}}^p + \tilde{\delta}^{*,p}$ slightly away from the boundary by the minimal $\|\cdot\|_\infty$ distance of 10^{-p}

$$\|\tilde{\mathbf{z}}^p + \tilde{\delta}^{*,p} - \tilde{\mathbf{x}}^p\|_\infty = 10^{-p}. \tag{10}$$

We then verify whether $\tilde{\mathbf{x}}^p$ and $\tilde{\epsilon}^p$ fulfill Property (4) but not Property (3). In such a case, we identify the vulnerability otherwise we choose a new $\tilde{\mathbf{z}}^p$ and repeat the process.

To find $\tilde{\delta}^{*,p}$, we exploit a standard geometric property of linear separators; specifically, $\tilde{\mathbf{w}}^{*,p}$ is perpendicular to the separator $\tilde{\mathcal{O}}^p(\tilde{\mathbf{x}}^p) = 0$. Thus, we seek $\tilde{\delta}^p$ by looking for $\alpha \in [0, \infty)$ such that

$$\tilde{\delta}^p : \tilde{\delta} = -\alpha y \tilde{\mathbf{w}}^{*,p}, \tag{11}$$

then using the bisection method to satisfy Property (9). The convergence of the bisection method is guaranteed by the fact that sufficiently large values of α imply that

$$y \neq \tilde{f}(\tilde{\mathbf{z}}^p + \tilde{\delta}^p) = \tilde{f}^p(\tilde{\mathbf{z}}^p + \tilde{\delta}^p), \tag{12}$$

since moving far enough along the direction perpendicular to the separator will eventually cross it, thereby changing the label. Conversely, as mentioned previously, it may be the case that no such $\tilde{\delta}^{*,p}$ satisfying Property (9) exists for a given $\tilde{\mathbf{z}}^p$, in which case we select a different $\tilde{\mathbf{z}}^p$ and repeat the procedure.

In order to find $\tilde{\mathbf{x}}^p$ satisfying Property (10), we again rely on Eq. (12) and obtain

$$\tilde{\mathbf{x}}^p = \tilde{\mathbf{z}}^p + \tilde{\delta}^{*,p} + y\, 10^{-p}\, \mathbf{sign}(\tilde{\mathbf{w}}^{*,p}). \tag{13}$$

Essentially, we project onto the surface of the ℓ_∞ ball of radius 10^{-p} around $\tilde{\mathbf{z}}^p + \tilde{\delta}^{*,p}$, moving a distance 10^{-p} away from the separator (in the direction consistent with the true label) along $\tilde{\mathbf{w}}^{*,p}$.

Algorithm 1: Precision-based attack.

Input: $\tilde{\mathbf{z}}^p$, p, η, $\tilde{f}$, and $\tilde{f}^p$
/* $\tilde{f}$ and $\tilde{f}^p$ only differ on the implementation, input and parameters are the same and represented at p fixed point precision */
/* $\eta = y\tilde{\mathbf{w}}^{*,p}$ for Linear Binary Classification */
/* $\eta = \tilde{W}_y^{*,p}$ for Linear Multiclass Classification */
/* $\eta = y\frac{\xi}{\|\xi\|_2}$ for Nonlinear Binary Classification */
/* $\eta = \frac{\zeta}{\|\zeta\|_2}$ for Nonlinear Multiclass Classification */
Output: $\tilde{\mathbf{x}}^p$ and $\tilde{\epsilon}^p$

$\alpha = 10^{-p}$;
do
  $\alpha = 2\alpha$; $\tilde{\delta}^p : \tilde{\delta} = -\alpha\eta$;
  if α *too large* **then** // Only for the nonlinear case
    return (**null**, **null**) // Attack Unsuccessful;
  end
while $y = \tilde{f}(\tilde{\mathbf{z}}^p + \tilde{\delta}^p)$ **or** $y = \tilde{f}^p(\tilde{\mathbf{z}}^p + \tilde{\delta}^p)$;
$\alpha_{\text{low}} = 0$; $\alpha_{\text{up}} = \alpha$; $\tilde{\delta}^p_{\text{old}} = \mathbf{0}$;
while *true* **do**
  $\alpha = \frac{\alpha_{\text{up}} + \alpha_{\text{low}}}{2}$; $\tilde{\delta}^p : \tilde{\delta} = -\alpha\eta$;
  if $y = \tilde{f}(\tilde{\mathbf{z}}^p + \tilde{\delta}^p) \neq \tilde{f}^p(\tilde{\mathbf{z}}^p + \tilde{\delta}^p)$ **then**
    break;
  end
  if $\tilde{\delta}^p_{old} = \tilde{\delta}^p$ **then**
    return (**null**, **null**) // Attack Unsuccessful;
  end
  $\tilde{\delta}^p_{\text{old}} = \tilde{\delta}^p$;
  if $y = \tilde{f}(\tilde{\mathbf{z}}^p + \tilde{\delta}^p)$ **then**
    $\alpha_{\text{low}} = \alpha$;
  else
    $\alpha_{\text{up}} = \alpha$;
  end
end
$\tilde{\mathbf{x}}^p = \tilde{\mathbf{z}}^p + \tilde{\delta}^p + 10^{-p}\mathbf{sign}(\eta)$; $\tilde{\epsilon}^p = 10^{-p}$;
return $(\tilde{\mathbf{x}}^p, \tilde{\epsilon}^p)$;

It is possible to generalize the previously described attack to the linear multiclass classification problem [39]. In this scenario, the output space is defined as $\mathcal{Y} = \{1, \ldots, c\}$. Accordingly, our model $f : \mathcal{X} \to \mathcal{Y}$ can then be written as

$$\hat{y} = f(\mathbf{x}) = \arg\max_{i \in \mathcal{Y}} \boldsymbol{\mho}_i(\mathbf{x}) = \arg\max_{i \in \mathcal{Y}} (W_i' \mathbf{x}), \tag{14}$$

where $\hat{y} \in \mathcal{Y}$, and $W \in \mathbb{R}^{d \times c}$ represents the trainable parameters. Here, W_i denotes the i-th column of W, and $\boldsymbol{\mho} : \mathcal{X} \to \mathbb{R}^c$ with $\boldsymbol{\mho}_i$ representing the i-th component of $\boldsymbol{\mho}$. A multiclass problem can be addressed in various ways. In this work, we employ the One Versus Rest (OVR) approach [39], which trains one binary classifier per class, assigning $y = +1$ to that class and $y = -1$ to all others. At prediction time, each classifier produces a score, and the class with the highest score is selected. Following the same principles and definitions introduced for binary classification, we can then state that

$$W_i^* = \arg\min_{\mathbf{w} \in \mathbb{R}^d} \|X\mathbf{w} - \mathbf{y}_y\|_2^2 + \lambda\|\mathbf{w}\|_2^2 = (X'X + \lambda I)^{-1} X' \mathbf{y}_i,$$
$$W_i^* = {}^{W_i^*}\!/_{\|W_i^*\|_2}, \quad \forall i \in \mathcal{Y}, \tag{15}$$

where, using the Iverson bracket notation, we represent $\mathbf{y}_i$ as follows

$$\mathbf{y}_i = 2[[y_1 = i], \cdots, [y_n = i]]' - 1. \tag{16}$$

Note that also the solution to Problem (15) can only be obtained through iterative numerical methods. Hence, for this setting, we define $\tilde{\theta} = \tilde{W}^*$, which is the `float64` solution to Problem (15). Then, we will also express it in its fixed-point representation with respect to p, denoted as $\tilde{\boldsymbol{\theta}}^p = \tilde{W}^{*,p}$.

In order to find $\tilde{\mathbf{x}}^p$ and $\tilde{\epsilon}^p$ that satisfy Property (4) but not Property (3), we follow the same procedure described for the binary classification problem. In fact, note that for multiclass classification, the model that yields the correct classification for $\tilde{\mathbf{z}}^p$ is given by $\tilde{\boldsymbol{\mho}}_y(\tilde{\mathbf{z}}^p)$ (i.e., $\tilde{W}_y^{*,p}$), and its associated binary label is always positive. Then, by simply setting $\eta = \tilde{W}_y^{*,p}$ in Algorithm 1, we obtain the generalization of our attack to multiclass linear classification.

B. Non-Linear Classification

Consider a binary classification problem involving a nonlinear classifier [3], [39]. Our model, $f : \mathcal{X} \to \mathcal{Y}$, can be represented in various forms. In this work, we adopt a neural network architecture featuring a single hidden layer with a ReLU activation function

$$\hat{y} = f(\mathbf{x}) = \text{sign}(\mho(\mathbf{x})) = \text{sign}(\mathbf{ReLU}(R\mathbf{x})'\mathbf{w}), \tag{17}$$

where $R \in \mathbb{R}^{r \times d}$, $\mathbf{w} \in \mathbb{R}^r$ are the learnable parameters and R is a hyperparameter that requires tuning [32]. Note that Classifier (17) is an universal approximator [3]. To learn the parameters, we follow the approach described in Section IV-A, with the exception of the algorithmic details. In fact, one may employ either end-to-end training using gradient-based optimization [3] or simpler methods that reduce the computational burden [19], [43]. In this work, we opt for the latter approach and obtain the optimal value of the parameters R, i.e., R^*, by sampling r vectors independently from a uniform distribution over the surface of the unit hypersphere in $\mathbb{R}^d$ stacking them row-wise in R^*. Then, we learn $\mathbf{w}^* \in \mathbb{R}^r$ using Ridge Regression (see Problem (7)) in the space induced by the rotation matrix R and the ReLU activation.

We define the parameter set $\tilde{\boldsymbol{\theta}} = \{\tilde{R}^*, \tilde{\mathbf{w}}^*\}$, which corresponds to the `float64` representations of R^* and $\mathbf{w}^*$. We express this set in its fixed-point representation with respect to p, denoted by $\tilde{\boldsymbol{\theta}}^p = \{\tilde{R}^{*,p}, \tilde{\mathbf{w}}^{*,p}\}$.

To obtain $\tilde{\mathbf{x}}^p$ and $\tilde{\epsilon}^p$ that satisfy Property (4), but not Property (3), we follow the same procedure as in the linear binary classification problem. The only difference lies in determining the direction that allows us to traverse the separator along the shortest path. A computationally efficient, exact solution for this problem does not exist, as it is NP-hard [30]. However, many effective and efficient heuristics have been proposed [2]. In this work, we exploit the simplest heuristic, namely using the derivative of $\tilde{\mho}$ computed at the point $\tilde{\mathbf{z}}^p$

$$\xi = \left.\frac{d\tilde{\mho}(\mathbf{x})}{d\mathbf{x}}\right|_{\mathbf{x}=\tilde{\mathbf{z}}^p} = \tilde{R}^{*,p}\text{diag}([\tilde{R}^{*,p}\tilde{\mathbf{z}}^p > 0])\tilde{\mathbf{w}}^{*,p}, \tag{18}$$

where $\text{diag}(\mathbf{v})$ denotes a diagonal matrix whose diagonal elements are the components of the vector $\mathbf{v}$. Note that more sophisticated approaches exist [2], but they are beyond the scope of our work. Finally, unlike the linear case discussed in Section IV-A, following this direction does not necessarily guarantee crossing the separator. In such instances, we select a different $\tilde{\mathbf{z}}^p$ and repeat the procedure. Then, by setting $\eta = y\frac{\xi}{\|\xi\|_2}$ in Algorithm 1, we obtain the generalization of our attack to binary nonlinear classification.

The extension to a nonlinear multiclass classifier [3], [39] follows directly from the results presented in this section and in Section IV-A. Similarly, the nonlinear binary classification model can be extended analogously to the linear case as

$$\hat{y}=f(\mathbf{x})=\arg\max_{i\in\mathcal{Y}} \boldsymbol{\mho}_i(\mathbf{x})=\arg\max_{i\in\mathcal{Y}}(\mathbf{ReLU}(R\mathbf{x})'W_i), \tag{19}$$

where we rely on the notation introduced earlier. Then we extend the approach by employing a OVR strategy and multiple ridge regression—analogous to the procedure for the multiclass linear classifier (see Problem (15)), but working in the feature space induced by the rotation matrix R and the ReLU activation. This formulation allows us to obtain the optimal parameters R^* and W^*.

During the learning phase, all computations are performed numerically. Thus, we define the parameter set $\tilde{\boldsymbol{\theta}} = \{\tilde{R}^*, \tilde{W}^*\}$, which corresponds to the `float64` representations of R^* and W^*. Furthermore, we express this set in its fixed-point representation with respect to p, denoted by $\tilde{\boldsymbol{\theta}}^p = \{\tilde{R}^{*,p}, \tilde{\mathbf{W}}^{*,p}\}$.

The final step is to define the direction ζ to be inserted into Algorithm 1. By employing the same argument exploited in multiclass linear classification (Section IV-A) and nonlinear binary classification, we can define ζ as

$$\zeta = \left.\frac{d\tilde{\boldsymbol{\mho}}_y(\mathbf{x})}{d\mathbf{x}}\right|_{\mathbf{x}=\tilde{\mathbf{z}}^p} = \tilde{R}^{*,p}\text{diag}([\tilde{R}^{*,p}\tilde{\mathbf{z}}^p > 0])\tilde{W}_y^{*,p}. \tag{20}$$

By setting $\eta = \frac{\zeta}{\|\zeta\|_2}$ in Algorithm 1, we obtain a generalization of our attack applicable to multiclass nonlinear classification.

C. Results

In this section, we present the results on the effectiveness of the attacks introduced in Sections IV-A and IV-B. We consider the MNIST dataset [26], focusing on binary classification using $\{0, 1\}$ and multiclass classification using $\{0, \ldots, 9\}$. Pixel values are normalized to the range $[-1, 1]$. For the binary case, we train on 100 randomly selected samples, while for the multiclass case we train on $10{,}000$ samples. These settings are applied to both the linear and nonlinear models. Training is performed with `float64` arithmetic, and hyperparameters are selected via cross-validation. In all experiments, $1{,}000$ mutually exclusive samples are used for testing. An additional set of 100 disjoint samples is employed to generate the attacks. The p fixed-point arithmetic is implemented by using `float64` arithmetic, but rounding every variable, both before and after each operation, to p-bit fixed-point precision. This approach provides the simplest yet most accurate setting, though one could consider more restrictive or even adversarial arithmetic implementations. We evaluate $p \in \{1, \ldots, 5\}$ overall, but generate attacks only at $p \in \{3, 4\}$, as these values are the smallest choices that do not substantially degrade the accuracy of the original models.

Table I reports the classifiers error percentage when implemented with `float64` arithmetic or with p fixed-point arithmetic while varying p. From Table I, we observe that

TABLE I: Classifiers error percentage.

Classifier	Classification error [%]					
	`float64`	$p=1$	$p=2$	$p=3$	$p=4$	$p=5$
Binary Linear	0.2	14.8	3.9	0.2	0.2	0.2
Multi Linear	15.1	89.1	54.9	15.4	15.0	15.1
Binary Nonlinear	0.3	48.1	0.3	0.3	0.3	0.3
Multi Nonlinear	7.3	86.2	10.8	7.2	7.3	7.3

TABLE II: Experimental assessment of the attack methodology on 100 samples.

Classifier	Successful Attacks	Verified instances α,β-CROWN	PyRAT	Marabou	NeVer2
$p=3$					
Binary Linear	25/100	25/25	25/25	25/25	25/25
Multi Linear	53/100	53/53	53/53	53/53	52/53
Binary Nonlinear	20/100	14/20	14/20	14/20	14/20
Multi Nonlinear	13/100	8/13	7/13	5/13	9/13
$p=4$					
Binary Linear	30/100	30/30	30/30	30/30	30/30
Multi Linear	41/100	39/41	39/41	39/41	38/41
Binary Nonlinear	20/100	19/20	19/20	19/20	19/20
Multi Nonlinear	16/100	11/16	11/16	11/16	12/16

$p = 2$ suffices for binary classification, and $p = 3$ for multiclass classification, to achieve accuracy comparable to the `float64` implementation. Additionally, as expected, nonlinear classifiers are more accurate when dealing with the more challenging multiclass setting.

Table II reports the effectiveness of the attacks in terms of the number of samples successfully compromised and the number of samples undetected by the top VNN-COMP verifiers (α,β-CROWN, PyRAT, and Marabou) as well as NeVer2, across varying p. From Table II, we observe that our heuristic attack is highly effective in evading detection: when the attack succeeds (i.e., Algorithm 1 does not return an unsuccessful result), it remains undetected in the majority of tested configurations.

To complete our presentation, Figure 2 illustrates a verified instance of successful attacks and the corresponding counterexample for the different classifiers at $p = 3$. From Figure 2, we notice a common trait among these "wild" samples: although the perturbation is imperceptible to the human eye, it still leads to issues in the verification process. Here we use "wild" in the sense of the work of [2].

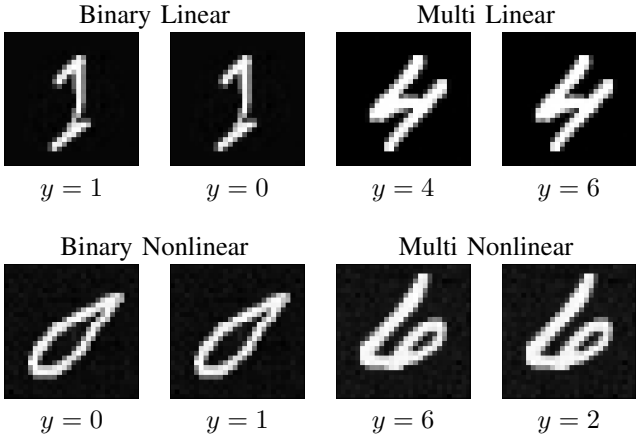


Fig. 2: Verified instance of successful attacks and associated counterexample for the different classifiers and $p = 3$.

V. Precision-Aware Verification

In this section, we present a modified version of NeVer2 that addresses the robustness issues discussed in Section IV, yielding a verifier that explicitly accounts for finite-precision effects and is resilient to such issues. Our implementation supports complete verification for linear classifiers (see Section IV-A) by leveraging *interval arithmetic* within the linear bounds propagation framework. In Section V-A, we introduce a novel abstraction for neural network representations, together with a verification procedure that maintains precision-dependent parameter intervals. Finally, in Section V-B, we apply our method to the data presented in Section IV-C, demonstrating that this approach successfully mitigates all identified attacks.

A. Methodology

To mitigate attacks that exploit how computations are performed at fixed-point precision p, we represent the linear model's boundary as the envelope of all possible implementations. This is achieved by incorporating *interval arithmetic* into the verification algorithm. Our interval representation is applied not only to the input values, but also to the model's weights and arithmetic operations. By doing so, our custom verifier integrated in pyNeVer can soundly account for all possible variations introduced by finite-precision arithmetic, providing a rigorous guarantee that the analyzed property holds against precision-based attacks.

a) Interval Arithmetic and Operations Definition: An *interval* U is defined by its inferior and superior limits, $\underline{a}$ and $\overline{a}$, as $U = [\underline{a}, \overline{a}] = \{\, a \in \mathbb{R} \mid \underline{a} \leq a \leq \overline{a} \,\}$. Let $U = [\underline{a}, \overline{a}]$ and $V = [\underline{b}, \overline{b}]$ be two intervals; we define the custom interval operations $U \oplus V := [\underline{a} + \underline{b}, \overline{a} + \overline{b}]$, $U \ominus V := [\underline{a} - \overline{b}, \overline{a} - \underline{b}]$, and $U \otimes V := \left[\min(\underline{ab}, \underline{a}\overline{b}, \overline{a}\underline{b}, \overline{a}\overline{b}),\ \max(\underline{ab}, \underline{a}\overline{b}, \overline{a}\underline{b}, \overline{a}\overline{b})\right]$. Interval matrices and vectors are objects whose elements are themselves intervals. For a matrix $M \in \mathbb{R}^{m \times n}$, its interval representation is denoted by $\langle M \rangle$, with elements $\langle M_{ij} \rangle = [\underline{M}_{ij}, \overline{M}_{ij}]$, $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$. Similarly, for a vector $v \in \mathbb{R}^n$ $\langle v_j \rangle = [\underline{v}_j, \overline{v}_j]$, $\quad j \in \{1, \ldots, n\}$. For brevity, we introduce a compact notation to define interval matrices and vectors from a standard matrix or vector. By writing $\langle \mathcal{Q} \rangle_\delta$, we denote a matrix or vector whose elements are intervals of the form $[q - \delta, q + \delta]$ where each element q corresponds to the respective element of the original matrix or vector, and $\delta > 0$ specifies a uniform bound applied to all elements. We define the positive and negative parts of an interval matrix $\langle M \rangle$ entry-wise as $\langle M^+ \rangle$ and $\langle M^- \rangle$:

$$\langle M_{ij}^+ \rangle = \begin{cases} [0, \overline{M}_{ij}], & \text{if } \underline{M}_{ij} \leq 0 \leq \overline{M}_{ij}, \\ [\underline{M}_{ij}, \overline{M}_{ij}], & \text{if } \underline{M}_{ij} \geq 0, \\ [0, 0], & \text{if } \overline{M}_{ij} \leq 0, \end{cases} \tag{21}$$

$$\langle M_{ij}^- \rangle = \begin{cases} [\underline{M}_{ij}, 0], & \text{if } \underline{M}_{ij} \leq 0 \leq \overline{M}_{ij}, \\ [0, 0], & \text{if } \underline{M}_{ij} \geq 0, \\ [\underline{M}_{ij}, \overline{M}_{ij}], & \text{if } \overline{M}_{ij} \leq 0, \end{cases} \tag{22}$$

where $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$. Matrix–vector, matrix–matrix multiplications and transposition are defined as in standard linear algebra, but using the custom interval operations $\oplus$ and $\otimes$ in place of addition and multiplication.

b) Interval Symbolic Bound Propagation: Our implementation of a complete verification algorithm integrated into NEVER2 combines the interval-based arithmetic described above with *symbolic interval propagation* (SIP) [4], [44] to compute exact bounds on the output layer, ensuring that our verifier provides correct answers even when operating in a setting where computations are performed at precision p.

Considering the property of interest 3, let $\mathbf{lb}$ and $\mathbf{ub}$ be two vectors such that their i-th components lb_i and ub_i are defined as $lb_i = \tilde{x}_i^p - \tilde{\epsilon}^p$ and $ub_i = \tilde{x}_i^p + \tilde{\epsilon}^p$. Let $\tau = \tilde{f}^p(\tilde{\mathbf{x}}^p)$ represent the *true label*, taking values in $\{-1, +1\}$ in the binary case, or in $\{1, \ldots, c\}$ in the multiclass case and let $\hat{y}_i$ denote the model output corresponding to class i.

In the binary case, the model (see 5) has a single output neuron, and the prediction is determined by the sign of $\mathbf{w}'\mathbf{x}$, i.e., the predicted label is either $+1$ or -1. Hence, the property of interest is satisfied if $\mathbf{w}^{'}\mathbf{x} > 0$ for all inputs where $\tau = +1$, and $\mathbf{w}^{'}\mathbf{x} < 0$ for all points where $\tau = -1$. By applying SIP, the minimum possible value of the single output neuron is

$$\langle \text{diff} \rangle = \left((\langle \mathbf{w} \rangle_\delta^+)' \otimes \langle lb \rangle_\delta \right) \oplus \left((\langle \mathbf{w} \rangle_\delta^-)' \otimes \langle ub \rangle_\delta \right) \quad (23)$$

where $\delta = 10^{-p}$. The value $\langle \underline{\text{diff}} \rangle$ is then multiplied by τ, and the property is violated if the result is strictly negative.

In the multiclass case (see 14), we must ensure that $\hat{y}_\tau - \hat{y}_l > 0$ for all $l \neq \tau$ and for all points $\mathbf{x}$ such that $lb_i \leq x_i \leq ub_i$. The property of interest is not satisfied if this condition is violated for some class $l \neq \tau$, that is, if there exist an input and $l \neq \tau$ such that $\hat{y}_l - \hat{y}_\tau \geq 0$. By applying SIP, the expression $\hat{y}_i - \hat{y}_\tau$ is abstracted as a linear function of the variables associated with the input, thereby accounting for all admissible inputs rather than a single realization [52]. The symbolic formula for the difference between $\hat{y}_l - \hat{y}_\tau$ is

$$\langle \Gamma_{l-\tau} \rangle = (\langle W \rangle_\delta^{'} \otimes e_l) \ominus (\langle W \rangle_\delta^{'} \otimes e_\tau), \quad (24)$$

where e_l and e_τ are one-hot vectors selecting the l-th and τ-th row, respectively. The numeric value is retrieved by the symbolic formula as

$$\langle \text{diff} \rangle = (\Gamma_{l-\tau}^+)' \otimes \langle lb \rangle_\delta \oplus (\Gamma_{l-\tau}^-)' \otimes \langle ub \rangle_\delta. \quad (25)$$

The property is violated if $\langle \underline{\text{diff}} \rangle \leq 0$ for some pair (l, τ).

B. Results

In this section, we report the results of using the modified version of NEVER2 described in Section V-A on the attack to linear NN classifiers described in Section IV-C which evades all the top VNN-COMP verifiers (α,β-CROWN, PYRAT, and MARABOU) as well as the original NEVER2.

Table III reports the evaluation of NEVER2 with interval-based abstraction using the same vulnerable examples obtained by the attacks in Section IV-C. In particular, we test instances that verifiers in Table II consider safe. In Table III we show the results of running NEVER2 with increasing values of p starting from 3, i.e., the precision of the first attack. Correctly, with $p = 3$ and $p = 4$ the verifier returns *Unsafe* for all the instances obtained with the same precision attack. This confirms that our methodology detects the implementation vulnerabilities. Increasing p, i.e., reducing the intervals size, the verifier starts to consider more and more instances to be safe because higher precision computations in NEVER2 do not detect adversaries that exist at lower precisions. Eventually, when $p = 7$, all instances are correctly declared safe.

TABLE III: Experimental results using NEVER2 with interval-based abstraction on the successful attacks of Table II.

Classifier	Verified instances				
	$p=3$	$p=4$	$p=5$	$p=6$	$p=7$
Attack with $p=3$					
Binary Linear	0/25	25/25	25/25	25/25	25/25
Multi Linear	0/53	17/53	51/53	53/53	53/53
Attack with $p=4$					
Binary Linear	0/30	0/30	30/30	30/30	30/30
Multi Linear	0/39	0/39	0/39	36/39	39/39

These experiments confirm that our mitigation approach is sound. Not only we can detect the attacks crafted with a specific precision, but we can also guarantee precision-aware safety. In particular, with the modified version of NEVER2 it is possible to verify robustness for a specific implementation precision and to determine the minimal precision for a linear NN classifier implementation to be robust.

VI. CONCLUSIONS

Our findings show that precision-based attacks pose a significant threat to current verification tools, including state-of-the-art methods. By augmenting verifiers with interval-based arithmetic, we offer a practical and effective strategy for ensuring consistency between verification and implementation, thus closing the gap introduced by precision discrepancies. Our empirical results demonstrate that this precision-aware approach successfully mitigates newly introduced attacks, thereby strengthening the reliability of certified guarantees for NN classifiers.

Nonetheless, our work is only a first step toward a comprehensive solution. A current limitation is that our approach applies solely to linear classifiers. We plan to bridge this gap by implementing an interval linear programming algorithm [8], [18], which will allow us to handle nonlinear classifiers as well. Moreover, the threat model can be further strengthened by incorporating mixed-precision representation and arithmetic and more potent adversarial capabilities, thereby offering a richer framework for precision-based attacks.

ACKNOWLEDGEMENTS

This work is partially supported by (i) project ELSA (European Lighthouse on Secure and Safe AI) funded by the European Union's Horizon Europe under the grant agreement No. 101070617, (ii) project SERICS (PE00000014) and FAIR (PE00000013) under the NRRP MUR program funded by the EU - NGEU, and (iii) project FISA-2023-00128 funded by the MUR program "Fondo italiano per le scienze applicate".

REFERENCES

[1] Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., Roli, F.: Evasion attacks against machine learning at test time. In: European conference in Machine learning and knowledge discovery in databases (2013)

[2] Biggio, B., Roli, F.: Wild patterns: Ten years after the rise of adversarial machine learning. Pattern Recognition **84**, 317–331 (2018)

[3] Bishop, C.M., Bishop, H.: Deep learning: Foundations and concepts. Springer Nature (2023)

[4] Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., Misener, R.: Efficient verification of relu-based neural networks via dependency analysis. In: AAAI Conference on Artificial Intelligence (2020)

[5] Brix, C., Bak, S., Johnson, T., Wu, H.: The fifth international verification of neural networks competition (VNN-COMP 2024): Summary and results. arXiv preprint arXiv:2412.19985 (2024)

[6] Brix, C., Müller, M.N., Bak, S., Johnson, T.T., Liu, C.: First three years of the international verification of neural networks competition (VNN-COMP). International Journal on Software Tools for Technology Transfer **25**(3), 329–339 (2023)

[7] Cherubin, S., Agosta, G.: Tools for reduced precision computation: a survey. ACM Computing Surveys **53**(2), 1–35 (2020)

[8] Chinneck, J.W., Ramadan, K.: Linear programming with interval coefficients. Journal of the operational research society **51**(2), 209–220 (2000)

[9] Demarchi, S., Guidotti, D., Pitto, A., Tacchella, A.: Formal verification of neural networks: A case study about adaptive cruise control. In: ECMS International Conference on Modelling and Simulation (2022)

[10] Demarchi, S., Guidotti, D., Pulina, L., Tacchella, A.: NeVer2: learning and verification of neural networks. Soft Computing **28**(19), 11,647–11,665 (2024)

[11] Girard-Satabin, J., Alberti, M., Bobot, F., Chihani, Z., Lemesle, A.: CAISAR: A platform for characterizing artificial intelligence safety and robustness. In: Workshop on Artificial Intelligence Safety (2022)

[12] Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: International Conference on Learning Representations (2015)

[13] Goyal, V., Das, R., Bertacco, V.: Hardware-friendly user-specific machine learning for edge devices. ACM Transactions on Embedded Computing Systems **21**(5), 1–29 (2022)

[14] Guidotti, D.: Verification of neural networks for safety and security-critical domains. In: Italian workshop on Planning and Scheduling (2022)

[15] Guidotti, D., Pandolfo, L., Pulina, L.: Leveraging satisfiability modulo theory solvers for verification of neural networks in predictive maintenance applications. Information **14**(7), 397 (2023)

[16] Guidotti, D., Pulina, L., Tacchella, A.: pyNeVer: A framework for learning and verification of neural networks. In: Automated Technology for Verification and Analysis (2021)

[17] Higham, N.J.: Accuracy and stability of numerical algorithms. SIAM (2002)

[18] Hladık, M.: Interval linear programming: A survey. Linear programming-new frontiers in theory and applications pp. 85–120 (2012)

[19] Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: theory and applications. Neurocomputing **70**(1-3), 489–501 (2006)

[20] Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: International conference on computer aided verification (2017)

[21] Jia, K., Rinard, M.: Exploiting verified neural networks via floating point numerical error. In: Static Analysis (2021)

[22] Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: International Conference on Computer Aided Verification (2017)

[23] Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljic, A., Dill, D.L., Kochenderfer, M.J., Barrett, C.W.: The marabou framework for verification and analysis of deep neural networks. In: International Conference on Computer Aided Verification (2019)

[24] Koren, I.: Computer arithmetic algorithms. AK Peters/CRC Press (2018)

[25] Kouvaros, P., Kyono, T., Leofante, F., Lomuscio, A., Margineantu, D.D., Osipychev, D., Zheng, Y.: Formal analysis of neural network-based systems in the aircraft domain. In: Formal Methods (2021)

[26] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)

[27] Lemesle, A., Lehmann, J., Le Gall, T.: Neural network verification with PyRAT. arXiv preprint arXiv:2410.23903 (2024)

[28] Leofante, F., Narodytska, N., Pulina, L., Tacchella, A.: Automated verification of neural networks: Advances, challenges and perspectives. arXiv preprint arXiv:1805.09938 (2018)

[29] Li, Y., Yu, Y., Liang, C., Karampatziakis, N., He, P., Chen, W., Zhao, T.: LoftQ: LoRA-Fine-Tuning-aware Quantization for large language models. In: International Conference on Learning Representations (2024)

[30] Marro, S., Lombardi, M.: Computational asymmetries in robust classification. In: International Conference on Machine Learning (2023)

[31] McDonald, G.C.: Ridge regression. Wiley Interdisciplinary Reviews: Computational Statistics **1**(1), 93–100 (2009)

[32] Oneto, L.: Model Selection and Error Estimation in a Nutshell. Springer (2020)

[33] Oneto, L., Ridella, S., Anguita, D.: Learning hardware-friendly classifiers through algorithmic stability. ACM Transactions on Embedded Computing Systems **15**(2), 1–29 (2016)

[34] Pitropakis, N., Panaousis, E., Giannetsos, T., Anastasiadis, E., Loukas, G.: A taxonomy and survey of attacks against machine learning. Computer Science Review **34**, 100,199 (2019)

[35] Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: International Conference on Computer Aided Verification (2010)

[36] Pulina, L., Tacchella, A.: NeVer: a tool for artificial neural networks verification. Annals of Mathematics and Artificial Intelligence **62**(3-4), 403–425 (2011)

[37] Saha, S.S., Sandha, S.S., Srivastava, M.: Machine learning for microcontroller-class hardware: A review. IEEE Sensors Journal **22**(22), 21,362–21,390 (2022)

[38] Sälzer, M., Lange, M.: Reachability is NP-complete even for the simplest neural networks. In: International Conference on Reachability Problems (2021)

[39] Shalev-Shwartz, S., Ben-David, S.: Understanding machine learning: From theory to algorithms. Cambridge university press (2014)

[40] Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: An abstract domain for certifying neural networks. ACM Programimming Languages **3**, 1–30 (2019)

[41] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R.: Intriguing properties of neural networks. In: International Conference on Learning Representations (2014)

[42] Tran, H.D., Yang, X., Lopez, D.M., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T.: NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: International Conference on Computer Aided Verification (2020)

[43] Vempala, S.S.: The random projection method. American Mathematical Soc. (2005)

[44] Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: neural information processing systems (2018)

[45] Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C., Kolter, J.Z.: Beta-CROWN: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In: Neural Information Processing Systems (2021)

[46] Wu, H., Isac, O., Zeljić, A., Tagomori, T., Daggitt, M., Kokke, W., Refaeli, I., Amir, G., Julian, K., Bassan, S., et al.: Marabou 2.0: a versatile formal analyzer of neural networks. In: International Conference on Computer Aided Verification (2024)

[47] Wu, H., Judd, P., Zhang, X., Isaev, M., Micikevicius, P.: Integer quantization for deep learning inference: Principles and empirical evaluation. arXiv preprint arXiv:2004.09602 (2020)

[48] Xiong, W., Lagerström, R.: Threat modeling-a systematic literature review. Computers & security **84**, 53–69 (2019)

[49] Xu, K., Shi, Z., Zhang, H., Wang, Y., Chang, K., Huang, M., Kailkhura, B., Lin, X., Hsieh, C.: Automatic perturbation analysis for scalable certified robustness and beyond. In: Neural Information Processing Systems (2020)

[50] 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 (2021)

[51] Zhang, H., Wang, S., Xu, K., Li, L., Li, B., Jana, S., Hsieh, C., Kolter, J.Z.: General cutting planes for bound-propagation-based neural network verification. In: Neural Information Processing Systems (2022)

[52] Zhang, H., Weng, T., Chen, P., Hsieh, C., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Neural Information Processing Systems (2018)

[53] Zhou, Y., Moosavi-Dezfooli, S.M., Cheung, N.M., Frossard, P.: Adaptive quantization for deep neural network. In: AAAI Conference on Artificial Intelligence (2018)

[54] Zombori, D., Bánhelyi, B., Csendes, T., Megyeri, I., Jelasity, M.: Fooling a complete neural network verifier. In: International Conference on Learning Representations (2021)