Neuron Similarity-Based Neural Network Verification via Abstraction and Refinement

Yuehao Liu[1] , **Yansong Dong**[2] , **Liang Zhao**[1] , **Wensheng Wang**[1] and **Cong Tian**[1,†]
[1]School of Computer Science and Technology, Xidian University
[2]Beijing Sunwise Information Technology Ltd
liuyh@stu.xidian.edu.cn, dongyansong0219@163.com, {lzhao, wswang}@xidian.edu.cn,
ctian@mail.xidian.edu.cn

Abstract

Deep neural networks (DNNs) have become integral to numerous safety-critical applications, necessitating rigorous verification of their trustworthiness. However, the problem of verifying DNNs has high computational complexity, and existing techniques have limited efficiency, insufficient to deal with large-scale network models. To address this challenge, we propose a novel abstraction-refinement verification method that reduces network size while maintaining verification accuracy. Specifically, the method quantifies the similarity between neurons based on various factors such as their interval outputs, and then merges similar neurons to generate a smaller abstract network. In addition, a counterexample-guided refinement process is developed to mitigate the impact of potential spurious counterexamples, so that verification results from the abstract network are applicable to the original network. We have implemented this method as a tool named ARVerifier and integrated it with three state-of-the-art verification tools for evaluation on ACAS Xu and MNIST benchmarks. Experimental results demonstrate that ARVerifier significantly reduces network size and yields verification time reductions by 11.61%, 18.70%, and 12.20% compared to α, β-CROWN, Verinet, and Marabou, respectively. Moreover, ARVerifier exhibits efficiency improvements by 26.64% and 46.87% compared to existing abstraction-refinement methods NARv and CEGAR-NN, respectively.

1 Introduction

Deep neural networks (DNNs) have been widely used in safety-critical domains owing to their exceptional performance, with applications spanning image recognition [Szegedy *et al.*, 2016], autonomous driving [Kiran *et al.*, 2021], and natural language processing [Wolf, 2020]. However, the inherent lack of robustness renders DNNs susceptible to adversarial attacks, wherein minor perturbations to otherwise correct inputs can precipitate significant errors [Athalye *et al.*, 2018; Zheng-Fei *et al.*, 2022; Yang *et al.*, 2023]. Consequently, prior to deployment in safety-critical environments, it is imperative to ensure that neural networks have expected output behavior, thus guaranteeing their reliability in these applications.

Given the imperative need for reliable AI systems in safety-critical domains, researchers have been actively exploring various approaches to enhancing the trustworthiness of DNNs. A particularly promising direction has emerged in the formal methods community, which develops verification techniques [Katz *et al.*, 2019; Henriksen and Lomuscio, 2020; Liu *et al.*, 2021; Kouvaros and Lomuscio, 2021; Lechner *et al.*, 2022] capable of automatically proving the satisfaction of specified input-output specifications for DNNs. However, DNN verification is generally NP-complete with respect to the size of the network model [Sälzer and Lange, 2022], which is computationally intensive even for simple specifications and networks. Consequently, despite recent advancements in verification techniques, the size of verifiable network models remains a significant limiting factor. This computational complexity necessitates the development of more efficient verification strategies to broaden the applicability of formal methods to larger-scale neural networks.

To address this challenge, we propose a novel abstraction-refinement verification method for DNNs based on neuron similarity. For efficient verification, the idea is to define similarity metric between neurons and merge similar neurons into one, so that the network N to be verified is abstracted into a smaller network $\bar{N}$. If $\bar{N}$ satisfies the specified safety property, it indicates that N also satisfies it. Conversely, if $\bar{N}$ violates the property, the verification process yields a counterexample cex and its genuineness is checked. In case cex is a genuine counterexample, it indicates that the original network N violates the safety property; otherwise cex is a spurious counterexample and the abstract network $\bar{N}$ is up to refinement. The refinement process increases both the size and accuracy of the abstract network to exclude the spurious cex, followed by iterative verification. This process adopts the counterexample guided abstraction refinement (CEGAR) paradigm [Clarke *et al.*, 2000], which is a promising technique for enhancing formal verification efficiency.

It is worth pointing out that the proposed method is generic and can be integrated with existing verification methods

†Corresponding author.

to enhance their efficiency. To evaluate the method, we implement it as a tool named ARVerifier and integrate it with state-of-the-art verifiers α, β-CROWN [Xu *et al.*, 2020; Wang *et al.*, 2021; Shi *et al.*, 2023], Verinet [Henriksen and Lomuscio, 2020], and Marabou [Katz *et al.*, 2019]. We apply ARVerifier to verify the safety properties of the ACAS Xu and MNIST benchmark datasets. Experimental results demonstrate that the method reduces verification time by up to 11.61%, 18.70%, and 12.20%, respectively, for these verifiers. Moreover, ARVerifier improves efficiency by 26.64% and 46.87%, respectively, compared to existing tools CEGAR-NN [Elboher *et al.*, 2020] and NARv [Liu *et al.*, 2024] that support structure-oriented CEGAR-based verification. The main contributions of this paper are summarized as follows.

- We propose an abstraction approach for DNNs that calculates neuron similarity by comprehensively considering interval, specific neuron value, weight, and layer depth influence. The approach minimizes accuracy loss in the network while reducing the number of identical neurons.
- We introduce a refinement approach that determines neuron restoration priority by thoroughly considering the values of neurons corresponding to counterexamples, along with their weights and layer-depth influence. The approach maximizes the restoration of network accuracy while increasing the number of identical neurons.
- We implement the proposed abstraction-refinement method as a tool named ARVerifier, and integrate it with three existing best-performed DNN verification engines: α, β-CROWN, Verinet, and Marabou. Extensive evaluations across multiple datasets demonstrate ARVerifier's advantage in verification efficiency.

2 Related Work

Formal verification of neural networks can be broadly categorized into two main approaches: complete and incomplete methods [Liu *et al.*, 2021]. Complete methods precisely encode the network, transforming the verification problem into a global optimization task. While these methods guarantee definitive results, they encounter significant scalability challenge when applied to larger networks. Conversely, incomplete verification methods provide improved scalability but may produce spurious counterexamples, resulting in a lack of guaranteed deterministic outcomes.

Complete methods typically employ rigorous encoding techniques such as mixed integer linear programming (MILP) and satisfiability modulo theory (SMT/SAT). Both MILP-based [Lomuscio and Maganti, 2017; Tjeng *et al.*, 2017; Weng *et al.*, 2018; Akintunde *et al.*, 2018; Botoeva *et al.*, 2020] and SAT/SMT-based [Ehlers, 2017; Liu *et al.*, 2021; Katz *et al.*, 2019; Jia *et al.*, 2023] approaches leverage solvers to address verification constraints. Recent studies [Henriksen and Lomuscio, 2020; Henriksen and Lomuscio, 2021; Wang *et al.*, 2021; Hashemi *et al.*, 2021] have introduced symbolic propagation and input refinement techniques to determine nonlinear neuron bounds within the network model. These output bounds serve as constraints for linear programming problems, effectively reducing the complexity introduced by nonlinear neurons during the verification. Most of these work focuses on deep neural networks (DNNs) with ReLU activation functions. While these methods offer completeness, they face scalability limitations. Our work addresses this scalability challenge by reducing the network size, thereby enhancing the applicability of these complete DNN verification techniques to larger networks.

Incomplete verification methods [Ma *et al.*, 2018; Pei *et al.*, 2017; Tian *et al.*, 2018; Yang *et al.*, 2022] encompass approaches that employ heuristic search or other dynamic analysis techniques to identify counterexamples that violate safety properties. While effective in finding violations, these methods cannot definitively prove that safety properties hold. Another category of incomplete methods, which includes our approach, leverages abstraction techniques. Existing abstract interpretation-based methods [Gehr *et al.*, 2018; Mirman *et al.*, 2018; Li *et al.*, 2019; Yang *et al.*, 2021] utilize abstract domains (e.g., interval, zonotope, and polyhedra) to approximate constraints from input to output layers, providing relatively accurate output range estimations. In contrast to these methods, the counterexample-guided abstraction and refinement (CEGAR) based approach employed in our work is founded on structural properties of DNNs rather than on computation.

Our work is closely related to CEGAR-NN [Elboher *et al.*, 2020] and NARv [Liu *et al.*, 2024]. CEGAR-NN introduces the structure-oriented CEGAR approach for neural network verification, but its effectiveness is limited due to a preprocessing step that quadruples the network size and an abstraction process that neglects input specifications. NARv improves upon this by only doubling the network size during preprocessing and considering input specifications in neuron merging. However, its neuron merging calculations remain insufficiently comprehensive, and the dependency graphs also constrain refinement flexibility. Unlike these works, our method is not a simple modification or extension but introduces key innovations in both abstraction and refinement processes. While adopting the same preprocessing step as theirs, our method considers not only neuron weights of the current layer but also specific inputs and the DNN hierarchy during abstraction, enabling a more comprehensive and accurate representation of neuron similarity. Furthermore, we employ an adversarial attack-based method for input set generation, effectively mitigating the impact of random generation on initial abstraction, which is a limitation in both previous methods. These enhancements, rooted in principled improvements rather than incremental adjustments, collectively contribute to improved efficiency and less accuracy loss, allowing for more effective verification of larger neural networks.

3 Preliminary

3.1 Deep Neural Network

A deep neural network N consists of $n+1$ layers: one input layer, $n-1$ hidden layers, and one output layer. Each layer i contains s_i neurons, with $v_{i,j}$ denoting the value of the j-th neuron. The output vector of the i-th layer is repre-

sented as $V_i = [v_{i,1}, \ldots, v_{i,s_i}]^T$. As the input x propagates through the network, each layer computes a weighted sum and applies an activation function. In this study, we focus on the rectified linear unit (ReLU) activation function, defined as $\mathrm{ReLU}(x) = \max(0, x)$. For each layer i ($i > 1$), a weight matrix $W_i \in \mathbb{R}^{s_i \times s_{i-1}}$ and a bias vector $B_i \in \mathbb{R}^{s_i}$ are associated. The value of the i-th layer is computed as:

$$V_i = \mathrm{ReLU}\left(W_i V_{i-1} + B_i\right) \tag{1}$$

This computation proceeds layer by layer until the network output $N(x) = V_n$ is obtained. The weight connecting neurons $v_{i-1,k}$ and $v_{i,j}$ is denoted as $w(v_{i-1,k}, v_{i,j})$. For each neuron $v_{i,j}$, we define its upper and lower bounds [Singh *et al.*, 2019] as $ub(v_{i,j})$ and $lb(v_{i,j})$, respectively, such that $lb(v_{i,j}) \leq v_{i,j} \leq ub(v_{i,j})$.

3.2 Neural Network Verification

Given a DNN N, an input constraint P, and an output constraint Q, the verification problem $\varphi = \langle N, P, Q \rangle$ aims to determine whether there is an input x satisfying P that leads to an output $y = N(x)$ satisfying Q, where P defines the input space and Q represents the negation of the desired safety property. This is to check whether there is a counterexample that violates the safety property. Similar to existing studies [Sälzer and Lange, 2022; Ruan *et al.*, 2018; Elboher *et al.*, 2020], we assume the input constraint is linear, the output layer contains a single neuron y, and the output constraint takes the form $y > c$ for a constant c. For verification problems $\varphi_1 = \langle N, P, Q \rangle$ and $\varphi_2 = \langle \bar{N}, P, Q \rangle$, $\bar{N}$ is an over-approximation of N if $N(x) \leq \bar{N}(x)$ for any x satisfying P. Consequently, if φ_2 is SAFE (UNSAT), then φ_1 is also SAFE (UNSAT), since $\bar{N}(x) \leq c \Rightarrow N(x) \leq c$. An input x satisfying P is a counterexample for N if $N(x) > c$. However, if x is a counterexample for $\bar{N}$ but $N(x) \leq c$, then x is called a spurious counterexample for N.

3.3 CEGAR-Based Neural Network Verification

Algorithm 1 specifies the general neural network verification method based on CEGAR. Given a verification problem $\langle N, P, Q \rangle$, the algorithm returns SAFE if the safety property is satisfied, or UNSAFE along with a counterexample cex.

To solve the verification problem $\langle N, P, Q \rangle$, the first step is to construct an abstract neural network $\bar{N}$ derived from the original network N with some abstraction approach. Usually, $\bar{N}$ is simpler than N so as to enhance verification efficiency. Lines 2-4 verify $\bar{N}$ using the $Verify$ function, which represents some verification method. If the result is SAFE, the algorithm returns SAFE, indicating that N is also safe. If the verification result is UNSAFE, the algorithm refines $\bar{N}$ through a loop (Lines 5-12). In each iteration, it extracts a counterexample cex and checks if it applies to the original network N. If so, it returns UNSAFE along with the counterexample. If cex is a spurious counterexample for N, it is used to refine $\bar{N}$, making the abstract network approximate the original network with cex no more a counterexample. If no applicable counterexample for N is found by the end of the loop, the algorithm returns SAFE, confirming that N is safe under the given conditions.

Algorithm 1 CEGAR-Based Neural Network Verification Algorithm

Input: $\langle N, P, Q \rangle$
Output: SAFE / UNSAFE

1: Build an abstract DNN $\bar{N}$ from N.
2: **if** $Verify(\bar{N}, P, Q)$ is SAFE **then**
3: **return** SAFE
4: **else**
5: **while** $Verify(\bar{N}, P, Q)$ is UNSAFE **do**
6: Extract counterexample cex.
7: **if** cex is a counterexample for N **then**
8: **return** UNSAFE, cex
9: **else**
10: Refine $\bar{N}$ using the cex.
11: **end if**
12: **end while**
13: **end if**
14: **return** SAFE

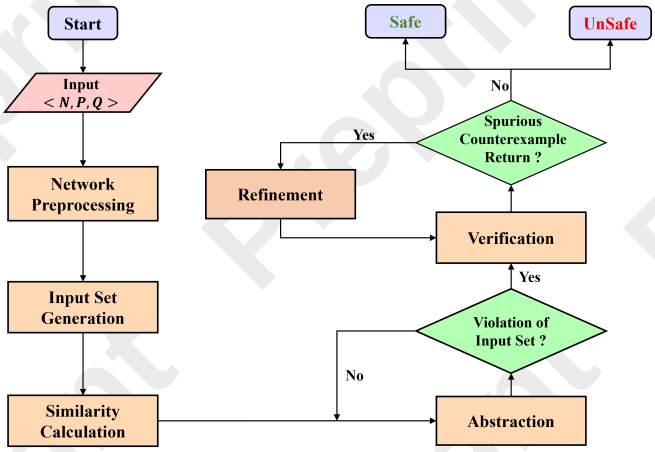


Figure 1: Similarity-Based Neural Network Verification Process

4 Methodology

To address the limitation of existing neural network verification techniques, we propose an abstraction-refinement verification method for DNNs based on neuron similarity, aiming to reduce the size of the verification network and thereby accelerate the verification process. This section provides details of the method, complementing and improving upon the key steps in Algorithm 1.

Specifically, the process of the verification method is shown in Figure 1. First, we preprocess the original network to be verified by classifying its neurons and converting it into an equivalent network to facilitate subsequent abstraction operations. Then, we employ adversarial search to generate an input set. This input set can be used to limit the number of abstraction operations and calculate neuron similarity. After that, by calculating the similarity between pairs of neurons of the same type, we select the most similar neurons and merge them into one neuron to reduce network size while minimizing accuracy loss. The neuron similarity calculation comprehensively considers various factors including interval similarity, value similarity under the same input, and merging impact

on the network. This merging process continues until it violates the constraints of the adversarially generated input set, resulting in an abstract network for verification. When a spurious counterexample appears, we determine which neuron to restore from the merged neurons by evaluating both the neuron's value corresponding to the counterexample's input and the impact of its restoration on the network. The proper merged neuron is restored and the network is refined, until a definitive verification result is obtained. Overall, by effectively reducing the network size through abstraction while preserving the model's critical behaviors and decision boundaries, the method can significantly enhance verification efficiency, particularly for complex neural networks. The key steps of the method are detailed as follows.

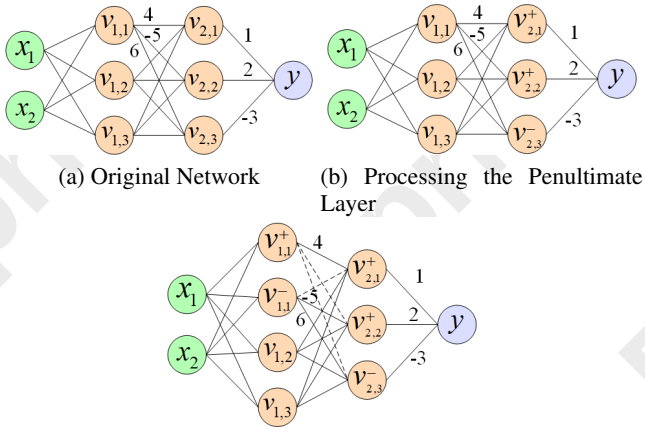
4.1 Network Preprocessing

The abstraction process aims to reduce the number of neurons by merging neurons within the same layer. To make this process feasible, we first preprocess the original network N. Specifically, we transform N into an equivalent network N', where equivalence means the same output for the same input. In N', neurons are classified into *inc* (increasing effect) and *dec* (decreasing effect) neurons. An *inc* neuron indicates that increasing its value leads to an increase in the network's final output, while a *dec* neuron has the opposite effect. Recall that the output layer consists of a single neuron. Neurons of the same type can be merged, and the resulting node retains the original classification.

The process of converting N into an equivalent network N' begins with designating the single output y of the last layer as the *inc* type, and then proceeds layer by layer backwardly. As illustrated in Figure 2, once all neurons in the $(i+1)$-th layer have been classified, each neuron $v_{i,j}$ in the i-th layer is split into an *inc* neuron $v^+_{i,j}$ and a *dec* neuron $v^-_{i,j}$, both inheriting the incoming edges of $v_{i,j}$. For outgoing edges, $v^+_{i,j}$ retains the positive weighted edges from $v_{i,j}$ to the *inc* neurons and the negative weighted edges to the *dec* neurons of the $(i+1)$-th layer, with the remaining outgoing edges set to 0. Conversely, $v^-_{i,j}$ retains the positive weighted edges from $v_{i,j}$ to the *dec* neurons and the negative weighted edges to the *inc* neurons of the $(i+1)$-th layer, with the other outgoing edges set to 0. The specific operations are defined by the following equations.

$$
\begin{aligned}
&w\left(v_{i-1,k}, v^+_{i,j}\right) = w\left(v_{i-1,k}, v^-_{i,j}\right) = w\left(v_{i-1,k}, v_{i,j}\right) \\
&w\left(v^+_{i,j}, v_{i+1,k}\right) = \begin{cases} w\left(v_{i,j}, v_{i+1,k}\right) & \text{if } w\left(v_{i,j}, v_{i+1,k}\right) \cdot \\ & \quad sign\left(v_{i+1,k}\right) > 0 \\ 0, & \text{otherwise} \end{cases} \\
&w\left(v^-_{i,j}, v_{i+1,k}\right) = \begin{cases} w\left(v_{i,j}, v_{i+1,k}\right) & \text{if } w\left(v_{i,j}, v_{i+1,k}\right) \cdot \\ & \quad sign\left(v_{i+1,k}\right) < 0 \\ 0 & \text{otherwise} \end{cases} \\
&sign\left(v_{i,j}\right) = \begin{cases} 1 & \text{if } v_{i,j} \text{ is } \textit{inc} \\ -1 & \text{otherwise} \end{cases}
\end{aligned}
\tag{2}
$$

Intuitively, when the value of the neuron $v^+_{i,j}$ increases, it affects the subsequent layer in two ways: it increases the values of all connected *inc* neurons and decreases the values of connected *dec* neurons. Both effects contribute to an increase in the final output y, which ensures $v^+_{i,j}$ is an *inc* neuron. On the contrary, $v^-_{i,j}$ is an *dec* neuron whose increase always leads to a decrease in the final output y.



(a) Original Network (b) Processing the Penultimate Layer

(c) Processing Neuron $v_{1,1}$

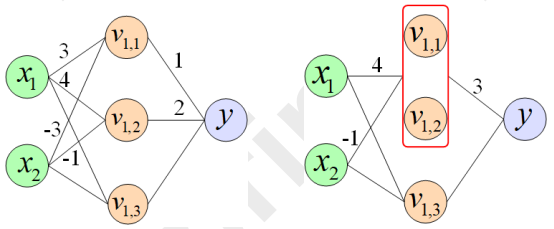
Figure 2: Construction of Initial Abstract Network

4.2 Input Set Generation

Prior to the abstraction phase, it is crucial to control the granularity of the initial abstraction. On the one hand, an overly aggressive abstraction with numerous abstracted neurons results in a coarse network, requiring too many refinement iterations to yield definitive verification results. On the other hand, insufficient abstraction fails to effectively reduce the network scale, potentially increasing verification time. The traditional method [Elboher *et al.*, 2020] for controlling abstraction granularity involves randomly generating an input set $X = \{x_1, \ldots, x_n\}$, in which all inputs satisfy the input constraint P and their corresponding outputs satisfy the output constraint Q. The abstraction process is terminated when an input in the set no longer satisfies the corresponding property after an iteration. However, this method's effectiveness is highly variable due to the random nature of the input set generation. To address this limitation, we propose to apply an adversarial attack method, projected gradient descent (PGD) [Madry *et al.*, 2017], to the original network N. The input set X is then formed using the inputs generated during the PGD iteration process. This method yields an input set closer to the decision boundary of the original network N, which is more effective in controlling the abstraction granularity and enhancing the quality of the initial abstraction. Specifically, the PGD-based method is expressed by the following equation:

$$
\begin{aligned}
x_0 &\sim \text{Uniform}(P) \\
x_{t+1} &= \Pi_P\left(x_t + \alpha \cdot \text{sign}(\nabla_x L(x_t, y))\right)
\end{aligned}
\tag{3}
$$

where x_0 is the initial input sampled from the input constraint P, x_t is the input at Iteration t, α is the step size, and Π_P denotes the projection onto P. The effectiveness of this method will be demonstrated in the experimental section.



(a) Initial Network Structure (b) Merging Neurons $v_{1,1}$ and $v_{1,2}$

Figure 3: Illustration of Neuron Merging

4.3 Abstraction

Having obtained an equivalent neural network N' from N, where each neuron is classified as either *inc* or *dec*, we perform a merging operation to construct an over-approximated network $\bar{N}'$. Consider two neurons $v_{i,j}$ and $v_{i,k}$ in the i-th layer, both classified as *inc*. Let $v_{i-1,p}$ and $v_{i+1,q}$ represent arbitrary neurons in the preceding and subsequent layers, respectively. The merging operation is specified as follows:

- delete neurons $v_{i,j}$ and $v_{i,k}$ along with their associated edges; and
- add a new neuron $v_{i,m}$ and assign the following values to the associated edges.

$$\begin{aligned}\bar{w}\left(v_{i-1,p}, v_{i,m}\right) &= \max(w\left(v_{i-1,p}, v_{i,j}\right), w\left(v_{i-1,p}, v_{i,k}\right)) \\ \bar{w}\left(v_{i,m}, v_{i+1,q}\right) &= w\left(v_{i,j}, v_{i+1,q}\right) + w\left(v_{i,k}, v_{i+1,q}\right)\end{aligned} \tag{4}$$

Intuitively, by employing the max operation in Equation (4), the newly formed neuron $v_{i,m}$ consistently produces an output greater than or equal to that of the original neurons $v_{i,j}$ and $v_{i,k}$ for any given input, as shown in Figure 3. Oppositely, when merging *dec* neurons, the max operation is replaced with the min operation.

A critical question that arises is how to select the optimal pair of neurons for merging. Our objective is to minimize the loss of network accuracy while reducing the network size. For this, we introduce the concept of *neuron similarity*. The similarity between two neurons, $v_{i,j}$ and $v_{i,k}$, is quantified by three metrics: interval similarity, specific value difference, and weight difference. The three metrics are expressed by the following equations.

$$\begin{aligned}\mathrm{IS}(v_{i,j}, v_{i,k}) &= \frac{\max\left(0, \min(ub_{i,j}, ub_{i,k}) - \max(lb_{i,j}, lb_{i,k})\right)}{\max(ub_{i,j}, ub_{i,k}) - \min(lb_{i,j}, lb_{i,k})} \\ \mathrm{SVD}(v_{i,j}, v_{i,k}) &= \sum_{x \in \mathrm{X}} |v_{i,j}(x) - v_{i,k}(x)| \\ \mathrm{WD}(v_{i,j}, v_{i,k}) &= \sum_{p} |w\left(v_{i-1,p}, v_{i,j}\right) - w\left(v_{i-1,p}, v_{i,k}\right)|\end{aligned} \tag{5}$$

Interval Similarity (IS) is defined as the ratio of the intersection length to the union length of two intervals, while these intervals are calculated by [Singh *et al.*, 2019]. The Specific Value Difference (SVD) is derived from the input set discussed in the previous section, calculating the difference between the output values of the two neurons for specific input values. The Weight Difference (WD) is computed as the sum of the weight differences between all neurons in the preceding layer and the merged neuron. The comprehensive similarity measure is expressed by the following equation.

$$\mathrm{NS}(v_{i,j}, v_{i,k}) = \frac{\mathrm{IS}(v_{i,j}, v_{i,k}) \cdot 2^i}{\mathrm{SVD}(v_{i,j}, v_{i,k}) \cdot \mathrm{WD}(v_{i,j}, v_{i,k})} \tag{6}$$

Our approach to neuron similarity considers multiple factors. Higher similarity is attributed to neurons that exhibit smaller differences in their values for the same input, have more similar input edge weights, and possess larger overlapping regions in their upper and lower bound intervals. Additionally, we factor in the impact of layer depth. As errors introduced by merging hidden layer neurons tend to be amplified through layer-by-layer propagation, the cumulative effect of multiple neuron errors can significantly influence the output of network. Considering this phenomenon, we incorporate a 2^i factor, which biases the merging process to proceed from the deeper layers towards the input. This strategy is not only more accurate compared to the simple backward layer-by-layer merging but can also reduce time consumption by decreasing the number of interval calculations. Overall, this comprehensive approach ensures a balance between abstraction efficiency and network accuracy, making it effective to verify complex neural networks.

4.4 Refinement

As illustrated in Algorithm 1, the initial abstraction yields an over-approximation network $\bar{N}'$ that may produce spurious counterexamples cex with $\bar{N}'(cex) > c$. To address this issue, we introduce a refinement process to eliminate spurious counterexamples, which is essentially the inverse of the abstraction. The process transforms $\bar{N}'$ into another approximation network $\bar{N}''$ such that $\bar{N}''(cex) \leq c$. Moreover, for any input x, the refinement ensures that $N(x) \leq \bar{N}''(x) \leq \bar{N}'(x)$, where N is the original network.

Specifically, the refinement operation involves selecting a neuron $v_{i,j}$ from the abstract neuron $\bar{v}_{i,j'}$ and reverting it to its pre-merged state, thereby reducing the network output according to a spurious counterexample cex. While abstraction aims to simplify the network, refinement seeks to restore network accuracy to eliminate spurious counterexamples, albeit at the cost of increasing the neuron count. The neuron selection in the refinement process is quantified by the following equation.

$$\begin{aligned}\mathrm{R}(v_{i,j}, \bar{v}_{i,j'}) = \sum_{p} \Big| &(\bar{v}_{i-1,p}(cex) \times (\bar{w}\left(v_{i-1,p}, \bar{v}_{i,j'}\right) \\ &- w\left(v_{i-1,p}, v_{i,j}\right)) \Big| \times \sum_{q} |\bar{w}\left(\bar{v}_{i,j'}, v_{i+1,q}\right) \\ &- w\left(v_{i,j}, v_{i+1,q}\right)| \times \frac{1}{2^i}\end{aligned} \tag{7}$$

Here, $\bar{v}_{i,j'}$ denotes the merged neuron, while $v_{i,j}$ represents a neuron in the initial network that was merged into $\bar{v}_{i,j'}$. The equation quantifies the impact on the network output when restoring $v_{i,j}$ from $\bar{v}_{i,j'}$. Since neurons preceding the i-th layer remain unaffected after the restoration, the product of the $(i-1)$-th layer's value in the counterexample cex

together with the weight difference of incoming edges before and after the restoration intuitively represents the impact of separating $v_{i,j}$. In addition, the weight difference of outgoing edges before and after the restoration affects the output of the current layer. Finally, similar to the abstraction, the refinement process considers the influence of layer depth. However, to maximize the impact of restored neurons on the output, we tend to prioritize the restoration of neurons in earlier layers.

5 Experimental Evaluation

The proposed method has been implemented as a tool named ARVerifier (Abstraction and Refinement Verifier for neural networks). The tool is generic as its backend verification engine is flexible, allowing integration with any complete verification tool capable of finding out counterexamples for unsafe verification problems. To assess the performance of ARVerifier, we integrate three state-of-the-art complete tools as the backend verification engines: α, β-CROWN, Verinet, and Marabou.

5.1 Benchmarks and Experimental Configuration

The neural networks considered in the experiments are from the fully connected benchmarks used in the international verification of neural networks competition (VNN-COMP) [Brix *et al.*, 2024; Brix *et al.*, 2023], specifically the ACAS Xu and MNIST datasets.

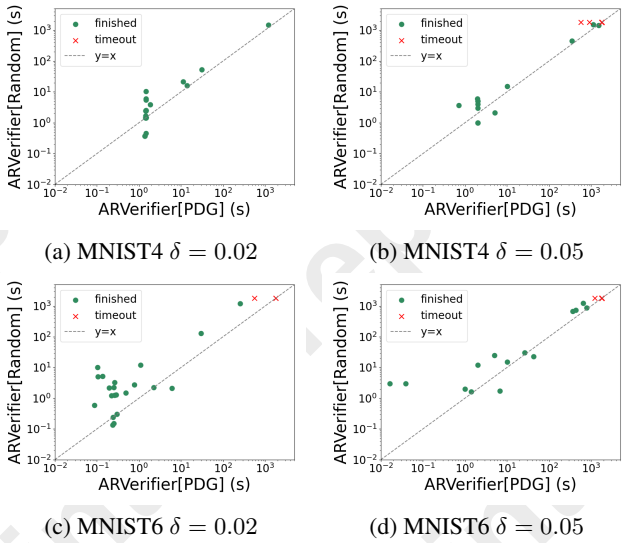
ACAS Xu [Julian *et al.*, 2016] consists of 45 ReLU-based DNNs for airborne collision avoidance, guiding aircraft steering based on sensor data. These networks, widely used in verification research, each have 5 inputs, 5 outputs, and 300 ReLU neurons in 6 layers. Our verification follows safety properties from [Katz *et al.*, 2017], testing 4 properties across all 45 networks and 6 on a single network.

The MNIST [LeCun, 1998] dataset consists of handwritten digits 0-9, represented as 28×28 grayscale images. We employ three fully connected DNNs: MNIST2, MNIST4, and MNIST6, respectively comprising 2, 4, and 6 hidden layers, each with 256 ReLU neurons. Our verification assesses local adversarial robustness, testing 25 images with l_∞ perturbations of 0.02 and 0.05.

Experiments are performed on a 64-bit Ubuntu 18.04 platform equipped with 64 GB of RAM and an Intel i7-7700 quad-core processor. All verification tools are implemented in Python 3.8. For the tools involving MILP solving, α, β-CROWN and Marabou use Gurobi 9.1, while Verinet uses Xpress 9.0 as the backend solver. Due to its feature, α, β-CROWN is executed with an additional NVIDIA TITAN RTX GPU with 24 GB memory. Besides, we impose a 30-minute timeout for each verification problem, while maintaining default values for other parameters.

5.2 Performance Enhancement: ARVerifier's Impact on Existing Verifiers

We first evaluate ARVerifier's efficacy in enhancing the performance of existing neural network verifiers. Specifically, three configurations of ARVerifier are implemented: ARVerifier[Marabou], ARVerifier[α, β-CROWN], and ARVerifier[Verinet], utilizing Marabou, α, β-CROWN, and Verinet as the backend verifiers, respectively. These configurations are then compared with their standalone counterparts without ARVerifier integration. Table 1 presents the experimental outcomes across six configurations for the ACAS Xu, MNIST2, MNIST4, and MNIST6 models, where ver denotes the number of verifiable safety properties, $t(s)$ represents the average verification time per property in seconds. Bold data highlight the superior results between each pair of compared configurations under the same benchmark. The ALL row at the bottom summarizes the total number of verifiable safety properties, the cumulative verification time across all the benchmarks.



(a) MNIST4 $\delta = 0.02$ (b) MNIST4 $\delta = 0.05$

(c) MNIST6 $\delta = 0.02$ (d) MNIST6 $\delta = 0.05$

Figure 4: Run Time Comparison (in Seconds, Log Scale) of ARVerifier on MNIST Models: PGD-Generated vs. Randomly Generated Input Sets

As is shown in Table 1, ARVerifier configurations generally outperform their original counterparts, solving more problems in less time. Specifically, ARVerifier[α, β-CROWN], ARVerifier[Verinet], and ARVerifier[Marabou] exhibit superior efficiency in verifying relatively larger network models MNIST4 and MNIST6 compared to the standalone α, β-CROWN, Verinet, and Marabou, respectively. However, for smaller models, ARVerifier[α, β-CROWN] and ARVerifier[Verinet] may require additional verification time. This is because these tools are already efficient in dealing with smaller models and can verify their problems in a short time. By contrast, ARVerifier's abstraction-refinement process and subsequent re-verification of refined networks introduce additional time overhead, ultimately leading to longer verification time for such cases. Another observation is that ARVerifier[Marabou] exhibits consistent improvement across most scenarios compared with the original Marabou, which also indicates the effectiveness of the proposed abstraction method in enhancing the efficiency of neural network verification. Overall, ARVerifier[α, β-CROWN], ARVerifier[Verinet], and ARVerifier[Marabou] reduce verification times by 11.61%, 18.70%, and 12.20%, respectively, compared to their original configurations.

Furthermore, we conduct ablation studies to evaluate the performance improvement achieved by using PGD adversar-

Model	Radius	ARVerifier[α, β-CROWN]		α, β-CROWN		ARVerifier[Verinet]		Verinet		ARVerifier[Marabou]		Marabou	
		ver	t(s)	ver	t(s)	ver	t(s)	ver	t(s)	ver	t(s)	ver	t(s)
ACAS Xu	-	186	9.52	186	**7.88**	186	10.82	186	**6.18**	**183**	**125.64**	180	147.45
MNIST2	0.02	25	2.18	25	**0.39**	25	3.21	25	**0.15**	25	58.69	25	**34.55**
	0.05	25	3.26	25	**0.31**	25	5.69	25	**1.45**	**25**	**522.65**	20	719.6
MNIST4	0.02	25	5.37	25	**1.26**	**25**	**50.37**	24	75.83	**23**	**240.96**	22	279.3
	0.05	**21**	**421.48**	19	497.07	**16**	**896.48**	11	1023.63	**11**	**1309.81**	7	1434.9
MNIST6	0.02	**25**	**51.75**	24	77.02	**23**	**176.31**	21	290.36	**16**	**884.26**	14	996.63
	0.05	**16**	**838.19**	12	941.30	**15**	**845.23**	10	1093.76	**5**	**1453.85**	4	1593.72
ALL		**323**	**34826.47**	316	39399.43	**315**	**51444.77**	302	63278.98	**288**	**135124.54**	272	153893.20

Table 1: Comparative Analysis of ARVerifier's Enhancements over α, β-CROWN, Verinet, and Marabou

Model	Radius	ARVerifier			NARv			CEGAR-NN		
		ver	t(s)	size	ver	t(s)	size	ver	t(s)	size
ACAS Xu	-	186	10.82	267	186	9.76	274	186	83.79	986
MNIST2	0.02	25	3.21	560	25	**2.87**	**533**	25	16.45	1834
	0.05	25	5.69	490	**25**	**3.46**	**463**	25	37.62	1743
MNIST4	0.02	**25**	**50.37**	**998**	24	82.11	1132	24	100.57	2453
	0.05	**16**	**896.48**	987	13	1107.56	**965**	10	1356.45	2489
MNIST6	0.02	**23**	**176.31**	**1484**	21	350.14	1684	21	314.85	4962
	0.05	**15**	**845.23**	1473	12	987.32	**1379**	10	1423.69	4627
ALL		**315**	**51444.77**	**6259**	306	65151.86	6430	301	96825.69	19094

Table 2: Comparison of CEGAR-based Methods on Verinet: ARVerifier, NARv, and CEGAR-NN

ial attacks for input set generation compared to random generation. We employ Verinet as the backend verifier for ARVerifier, varying only the input set generation method to isolate its impact. To obtain more pronounced results, we performed these experiments on the relatively larger MNIST4 and MNIST6 models. As illustrated in Figure 4, the majority of data points and crosses are positioned above the gray line, demonstrating that the PGD-based input set generation method outperforms random generation across most verification scenarios.

5.3 Comparative Analysis: ARVerifier vs CEGAR-based Approaches

We conduct a comparative analysis of ARVerifier against existing CEGAR-based approaches NARv [Liu *et al.*, 2024] and CEGAR-NN [Elboher *et al.*, 2020], all utilizing Verinet as their backend verification engine. In this analysis, α, β-CROWN is not considered because it lacks a unified solving strategy and requires manual fine-tuning to achieve optimal results for networks of different sizes and obtained with different abstraction strategies. Marabou is not used owing to its relatively slower verification speed. Table 2 presents the experimental outcomes for these three configurations across the ACAS Xu, MNIST2, MNIST4, and MNIST6 models. For each configuration, ver denotes the number of verifiable safety properties, $t(s)$ represents the average verification time per property in seconds, and $size$ indicates the average neuron count in the network following the initial abstraction. Bold data highlight the best results among compared configurations under the same benchmark. The ALL row summarizes the total number of verifiable safety properties, cumulative verification time, and aggregate neuron count after the initial abstraction across all benchmarks.

According to the results in Table 2, ARVerifier exhibits superior performance compared to NARv and CEGAR-NN across the key metrics: problem-solving quantity, verification time, and abstract network size. CEGAR-NN's poor network size reduction stems from its preprocessing stage that quadruples the network size, as well as its structure-based abstraction process that disregards input attributes. NARv matches ARVerifier in network size reduction, its use of randomly generated input sets for abstraction restriction leads to suboptimal initial abstraction, necessitating multiple refinements and resulting in limited verification performance. Specifically, ARVerifier demonstrates improvements by 26.64% and 46.87% in verification efficiency compared to NARv and CEGAR-NN, respectively. These results underscore the effectiveness of ARVerifier in neural network verification.

6 Conclusion

This paper proposes a generic abstraction-refinement method for neural networks verification. Specifically, we introduce an abstraction procedure based on neuron similarity and a corresponding refinement strategy to improve the verification efficiency. We implement the method in a tool and evaluate it on two widely used benchmarks. Experimental results demonstrate that our method can significantly enhance the scalability and efficiency of state-of-the-art complete verification tools without compromising their accuracy, particularly for larger network models.

Acknowledgments

This work is supported by National Natural Science Foundation of China through grant No.62192734, 62402372.

References

[Akintunde *et al.*, 2018] Michael Akintunde, Alessio Lomuscio, Lalit Maganti, and Edoardo Pirovano. Reachability analysis for neural agent-environment systems. In *Sixteenth international conference on principles of knowledge representation and reasoning*, 2018.

[Athalye *et al.*, 2018] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In *International conference on machine learning*, pages 284–293. PMLR, 2018.

[Botoeva *et al.*, 2020] Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. Efficient verification of relu-based neural networks via dependency analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3291–3299, 2020.

[Brix *et al.*, 2023] 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). *International Journal on Software Tools for Technology Transfer*, 25(3):329–339, 2023.

[Brix *et al.*, 2024] Christopher Brix, Stanley Bak, Taylor T Johnson, and Haoze Wu. The fifth international verification of neural networks competition (vnn-comp 2024): Summary and results. *arXiv preprint arXiv:2412.19985*, 2024.

[Clarke *et al.*, 2000] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification: 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000. Proceedings 12*, pages 154–169. Springer, 2000.

[Ehlers, 2017] Ruediger Ehlers. Formal verification of piecewise 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*, pages 269–286. Springer, 2017.

[Elboher *et al.*, 2020] Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. An abstraction-based framework for neural network verification. In *Computer Aided Verification: 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Proceedings, Part I 32*, pages 43–65. Springer, 2020.

[Gehr *et al.*, 2018] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2018.

[Hashemi *et al.*, 2021] Vahid Hashemi, Panagiotis Kouvaros, and Alessio Lomuscio. Osip: Tightened bound propagation for the verification of relu neural networks. In *Software Engineering and Formal Methods: 19th International Conference, SEFM 2021, Virtual Event, December 6–10, 2021, Proceedings 19*, pages 463–480. Springer, 2021.

[Henriksen and Lomuscio, 2020] Patrick Henriksen and Alessio Lomuscio. Efficient neural network verification via adaptive refinement and adversarial search. In *ECAI 2020*, pages 2513–2520. IOS Press, 2020.

[Henriksen and Lomuscio, 2021] Patrick Henriksen and Alessio Lomuscio. Deepsplit: An efficient splitting method for neural network verification via indirect effect analysis. In *IJCAI*, pages 2549–2555, 2021.

[Jia *et al.*, 2023] Fuqi Jia, Rui Han, Pei Huang, Minghao Liu, Feifei Ma, and Jian Zhang. Improving bit-blasting for nonlinear integer constraints. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 14–25, 2023.

[Julian *et al.*, 2016] Kyle D Julian, Jessica Lopez, Jeffrey S Brush, Michael P Owen, and Mykel J Kochenderfer. Policy compression for aircraft collision avoidance systems. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–10. IEEE, 2016.

[Katz *et al.*, 2017] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*, pages 97–117. Springer, 2017.

[Katz *et al.*, 2019] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In *Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I 31*, pages 443–452. Springer, 2019.

[Kiran *et al.*, 2021] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2021.

[Kouvaros and Lomuscio, 2021] Panagiotis Kouvaros and Alessio Lomuscio. Towards scalable complete verification of relu neural networks via dependency-based branching. In *IJCAI*, pages 2643–2650, 2021.

[Lechner *et al.*, 2022] Mathias Lechner, Đorđe Žikelić, Krishnendu Chatterjee, and Thomas A Henzinger. Stability verification in stochastic control systems via neural network supermartingales. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 7326–7336, 2022.

[LeCun, 1998] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[Li *et al.*, 2019] Jianlin Li, Jiangchao Liu, Pengfei Yang, Liqian Chen, Xiaowei Huang, and Lijun Zhang. Analyzing deep neural networks with symbolic propagation:

Towards higher precision and faster verification. In *Static Analysis: 26th International Symposium, SAS 2019, Porto, Portugal, October 8–11, 2019, Proceedings 26*, pages 296–319. Springer, 2019.

[Liu *et al.*, 2021] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, Mykel J Kochenderfer, et al. Algorithms for verifying deep neural networks. *Foundations and Trends in Optimization*, 4(3-4):244–404, 2021.

[Liu *et al.*, 2024] Jiaxiang Liu, Yunhan Xing, Xiaomu Shi, Fu Song, Zhiwu Xu, and Zhong Ming. Abstraction and refinement: Towards scalable and exact verification of neural networks. *ACM Transactions on Software Engineering and Methodology*, 33(5):1–35, 2024.

[Lomuscio and Maganti, 2017] Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351*, 2017.

[Ma *et al.*, 2018] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*, pages 120–131, 2018.

[Madry *et al.*, 2017] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

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

[Pei *et al.*, 2017] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18, 2017.

[Ruan *et al.*, 2018] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. *arXiv preprint arXiv:1805.02242*, 2018.

[Sälzer and Lange, 2022] Marco Sälzer and Martin Lange. Reachability in simple neural networks. *Fundamenta Informaticae*, 189(3-4):241–259, 2022.

[Shi *et al.*, 2023] Zhouxing Shi, Qirui Jin, J Zico Kolter, Suman Jana, Cho-Jui Hsieh, and Huan Zhang. Formal verification for neural networks with general nonlinearities via branch-and-bound. 2023.

[Singh *et al.*, 2019] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, 2019.

[Szegedy *et al.*, 2016] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[Tian *et al.*, 2018] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*, pages 303–314, 2018.

[Tjeng *et al.*, 2017] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

[Wang *et al.*, 2021] 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. *Advances in Neural Information Processing Systems*, 34:29909–29921, 2021.

[Weng *et al.*, 2018] Lily Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. Towards fast computation of certified robustness for relu networks. In *International Conference on Machine Learning*, pages 5276–5285. PMLR, 2018.

[Wolf, 2020] Thomas Wolf. Transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2020.

[Xu *et al.*, 2020] 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. *arXiv preprint arXiv:2011.13824*, 2020.

[Yang *et al.*, 2021] Pengfei Yang, Renjue Li, Jianlin Li, Cheng-Chao Huang, Jingyi Wang, Jun Sun, Bai Xue, and Lijun Zhang. Improving neural network verification through spurious region guided refinement. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 389–408. Springer, 2021.

[Yang *et al.*, 2022] Xiao Yang, Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu. Boosting transferability of targeted adversarial examples via hierarchical generative networks. In *European Conference on Computer Vision*, pages 725–742. Springer, 2022.

[Yang *et al.*, 2023] Yuting Yang, Pei Huang, Juao Cao, Feifei Ma, Jian Zhang, and Jintao Li. Quantifying robustness to adversarial word substitutions. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 95–112. Springer, 2023.

[Zheng-Fei *et al.*, 2022] Yu Zheng-Fei, Yan Qiao, and Zhou Yun. A survey on adversarial machine learning for cyberspace defense. *Journal of Automation*, 48(07):1625–1649, 2022.