VeRe: Verification Guided Synthesis for Repairing Deep Neural Networks

Document Version
Accepted author manuscript

Link to publication record in Manchester Research Explorer

Citation for published version (APA):

Published in:
46th International Conference on Software Engineering (ICSE 2024)

Citing this paper
Please note that where the full-text provided on Manchester Research Explorer is the Author Accepted Manuscript or Proof version this may differ from the final Published version. If citing, it is advised that you check and use the publisher's definitive version.

General rights
Copyright and moral rights for the publications made accessible in the Research Explorer are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Takedown policy
If you believe that this document breaches copyright please refer to the University of Manchester’s Takedown Procedures [http://man.ac.uk/04Y6B0] or contact uml.scholarlycommunications@manchester.ac.uk providing relevant details, so we can investigate your claim.
VeRe: Verification Guided Synthesis for Repairing Deep Neural Networks

Anonymous Author(s)*

1 INTRODUCTION

Over the past decade, deep neural networks (DNNs) have brought about breakthroughs in many areas such as computer vision [3], natural language processing [9, 19], and speech recognition [75]. However, the adoption of DNNs in safety-critical domains has been slow due to concerns about their dependability. For instance, adversaries can manipulate the input data in a way that is imperceptible to humans but can cause the model to make incorrect decisions. Such vulnerability leads to serious safety concerns in applications such as autonomous vehicles [8] and medical diagnosis [74]. While DNNs are promising to revolutionize safety-critical domains (to some extent) as well, it is yet crucial to ensure that the DNN models are developed and deployed with safety and reliability requirements in mind so as to fully realize their potential.

The repair of DNNs refers to the activity of fixing the 'bugs' of a neural network by modifying its architecture or parameters. Such bugs can arise from multiple sources which have been extensively studied in different communities including training errors [11], adversarial attacks [14, 28], backdoor attacks [48] and distribution drift [68, 82]. To counter these potential risks, periodic and efficient repair of DNNs is essential in DNN deployment to ensure their performance and reliability in practice. Classical methods in machine learning for DNN repair include adversarial training [25, 26, 70], input sanitization, fine-tuning, transfer learning [17, 85], data augmentation [49, 58, 86], etc. However, these methods are highly data-driven, which often have poor performance when the available 'bugged' data is scarce or of low quality. To address the issue, neuron-level error localization and fixing methods have been recently proposed in the software engineering community for more effective and efficient DNN repair. For example, Arachne [64] uses gradients and activation values to identify problematic neurons, and then utilizes the differential evolution algorithm to generate patches that can repair the neural network model. RM [32], on the other hand, estimates the impact of neurons on both positive and negative samples by leveraging gradients. When repairing the neural network, RM fine-tunes neurons that have greater impact. CARE [67] incorporates a causal model to analyze the cause-and-effect relationship between neurons and inaccurate behavior, and utilizes the Particle Swarm Optimization (PSO) algorithm to generate neuron-level patches for repairing the model.

As for neuron-level DNN repair, the fundamental technical challenge is how to understand the behavior of large amount of neurons in the DNNs on the data samples (especially when they exhibit erroneous behaviors), and furthermore how to locate a small number of responsible neurons for fixing. While previous gradient-based or causality-based approaches are effective in some cases, they are in general statistics-based and thus heavily rely on the availability of massive data samples for larger DNNs to be effective (e.g., for CARE [67], 20,000 samples are needed to achieve 94.70% generalization for fully-connected model on ACAS Xu dataset). This can be problematic for scenarios when the data owner does not want to share much data for the repair task, or data collection is too expensive and there are simply not enough faulty data2. Besides, the different kinds of heuristics in terms of layer or neuron selection hinder their easy adoption over a wide range of neural network repair tasks. Formal verification uses mathematical methods to rigorously prove whether a system or software meets its intended specification under all possible scenarios [7]. Several common verification techniques like [22, 26, 39, 40, 44, 57, 62, 63] have been adapted for DNNs to model or abstract their behaviors and furthermore

*We use 'bugs' to denote different kinds of inputs that could trigger an error in the model's output.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICSE 2024, April 2024, Lisbon, Portugal
© 2023 Association for Computing Machinery.
ACM ISBN 978-1-x-xxxx-xxxx-1/TY/MMM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2while VeRe is shown to be effective in presence of both massive or scarce data.
verify them against safety properties [51] and others [14, 31]. In particular, a sound abstraction of a DNN model can be effectively used to characterize how the DNN’s behavior would change upon applying perturbation on the input. The implication is that, DNN verification is naturally connected with neuron-level DNN repair which essentially requires to measure the significance of a neuron (either before or after the repair) on the model’s erroneous behavior by applying perturbation on a neuron and observe the model’s output change. If a small perturbation on a neuron leads to a large output change, such a neuron is considered more responsible for the error. One step further, repairing the neuron (ruling out the errors) can be formalized as an optimization problem to search for the parameters minimizing the output change upon perturbation on those responsible neurons. Notice that unlike some existing works [49, 87] that use a loss function for optimization in the repair step, our method does not ‘over-learn’ the samples that have been correctly classified, thus avoiding the overfitting problem (details later).

To realize the above idea, we propose VeRe (Verification Guided Synthesis for Repairing Deep Neural Networks), a novel verification guided synthesis framework for repairing the violation of safety properties and backdoor attacks in DNNs. VeRe follows the classical procedure of neuron-level error localization and error fixing while both steps are guided by reachability analysis in the form of linear approximation provided by formal verification. The linear approximation can soundly measure how significant the repair of a neuron contributes to fixing the behavior of negative samples and provide the target behavior as guidance in the fixing step. Fig. 1 shows the details. Technically, we employ the verification tool CROWN [83] to establish a linear approximation of the activation of a neuron in a fully connected layer, and figure out the target behavior of this activation via the linear approximation after this layer. The repair significance of this neuron for the current negative sample is the maximum improvement of the difference between the scores of the correct and the misled classification of this sample, and its total repair significance is the sum of that for each negative sample. By sorting the repair significance, we decide the order of repair from the highest to the lowest. The target behavior, along with the positive samples, is used for guiding the fixing of this neuron by constructing an optimization problem looking for repair that enforces the neuron behavior into the target with the smallest turbulence on positive samples. The procedure of such error localization and error fixing is conducted iteratively until all the negative samples are successfully repaired or the number of iterations reaches a threshold. In this workflow, formal verification plays an important role in abstracting behaviors of DNNs and neurons, measuring the repair significance of each neuron, and calculating the target behavior as repair guidance.

2 BACKGROUND

Deep neural network. In this work, we focus on DNNs for classification tasks. A DNN is a function $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ which assigns a high-dimensional input $x \in \mathbb{R}^m$ to an output $f(x) \in \mathbb{R}^n$. A classification DNN chooses the dimension with the highest score as the classification result, i.e., its classification behavior can be described as $C_f(x) = \arg \max_{1 \leq i \leq n} f(x)_i$. A DNN usually contains multiple hidden layers, such as convolutional layers, pooling layers, activation layers, etc., and the behavior of a DNN $f$ is the composition of the functions between layers sequentially from the first layer (input layer) to the last (output layer), i.e., $f = f_{l-1} \circ \cdots \circ f_1$, where $l$ is the total number of layers in $f$, and $f_i$ is the function from the $i$-th layer to the next. In particular, we use $f^l$ to denote the subnetwork from the $l$-th layer (as the input layer) to the output layer of the neural network $f$, i.e., $f^l = f_{l-1} \circ \cdots \circ f_1$ and $h^l$ to denote the subnetwork from the input layer to the $l$-th layer (as the output layer), i.e., $h^l = f_{l-1} \circ \cdots \circ f_1$. The output of each neuron, except those in the input layer, is obtained by the corresponding transformation of the relevant nodes in the previous layer, usually in the form of the composition of an affine transformation and a non-linear activation function. Formally, for a hidden neuron $j$ in layer $i$, its output $h_{ij}$ with respect to input $x$ can be calculated as:

$$h_{ij}(x) = \sigma(w_{ij} \cdot h^{i-1}(x) + b_{ij})$$

with the activation function $\sigma$, the in-weights $w_{ij}$, and the bias $b_{ij}$.

Safety properties and backdoor attacks. Safety properties refer to those descriptions that bad things never happen. In the setting of DNNs, a safety property requires that a DNN should behave correctly in a given input set. The property violation rate (VR) of a safety property measures how much the property is violated in the input space w.r.t. a pre-defined probability measure $\mathbb{P}$.

Definition 2.1. Let $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ be a DNN, $X \subseteq \mathbb{R}^m$ an input space, and $P \subseteq \mathbb{R}^n$ an output property: A safety property is a tuple $\varphi = (f, X, P)$, and it holds iff for any $x \in X$, $f(x) \in P$. The property violation rate (VR) of a safety property $\varphi = (f, X, P)$ is defined as $VR(\varphi) = \mathbb{P}(f(x) \notin P \mid x \in X)$.

Backdoor attacks occur when an attacker implants a hidden trigger in the DNN that can be activated to cause it to make incorrect predictions. This trigger may be added with access to the training input set to trigger the attack.
data or model architecture. Similarly, the attack success rate (SR) of a backdoor attack reflects its effectiveness.

**Definition 2.2.** Let \( f : \mathbb{R}^m \rightarrow \mathbb{R}^n \) be a backdoored DNN, and \( X \subseteq \mathbb{R}^m \) an input set with the trigger. The attack success rate (SR) on \( X \) is defined as \( \text{SR}(f, X) = \mathbb{P}(C_f(x) = t \mid x \in X) \), where \( t \) is the target label of backdoor.

**Linear relaxation.** Linear relaxation, also called linear approximation, is an important technique in the field of formal verification of DNNs. It can establish linear bounds between layers and compute provable reachability analysis on output neurons, which is widely used in neural network verification and certified adversarial defense [52, 88]. Here we briefly introduce how CROWN conducts linear relaxation. The key to linear relaxation is to over-approximate the behavior of non-linear activation functions with two affine functions as lower/upper bound. In CROWN, linear relaxation is designed in a symbolic way, so that the linear bound can propagate. For example, a ReLU activation \( z^* = \text{ReLU}(z) \) with \( z \in [l, u] \) \((l < 0 < u)\) can be linearly relaxed to \( \frac{u - l}{u - l} \leq z^* \leq \frac{u(l - z)}{u - l} \), and the numerical lower bound of \( z^* \) can be obtained by substituting \( z \) in the term \( \frac{u(l - z)}{u - l} \) with its affine lower bound, and iteratively substituting each new variable with one of its affine bounds according to its coefficient, until all the variables in the term are input variables.

In this way, given a region \( \Delta \subseteq \mathbb{R}^m \), usually in the form of a high-dimensional interval, a DNN \( f \) can be linearly relaxed to its lower bound \( g^{\text{lower}}(x) = w^T x + b \) and upper bound \( g^{\text{upper}}(x) = \overrightarrow{w}^T x + \overrightarrow{b} \) satisfying

\[
g^{\text{lower}}(x) \leq f(x) \leq g^{\text{upper}}(x), \forall x \in \Delta. \tag{2}
\]

**Problem formulation.** Our neural network repair problem can be formally defined as follows:

Given a classification DNN \( f : \mathbb{R}^m \rightarrow \mathbb{R}^n \) with a set of inputs \( D \subseteq \mathbb{R}^m \), partitioned into the misclassified \( D_m \) and the correctly classified \( D_c \), we need to synthesize a repaired DNN \( f' \), which differs from \( f \) only in the weights and biases, so that the erroneous behaviors in \( D_m \) are fixed as much as possible while the accuracy of the model is maintained.

Remark that the training data are unavailable throughout the repair. The limited samples in \( D \) for repair are collected through adding backdoor trigger or sampling, which is the case in real-world testing scenarios.

### 3 VERIFICATION GUIDED REPAIR SYNTHESIS

In this section, we present VeRe, a novel verification based repair approach for neural networks. An overview is presented in Fig. 1. VeRe has two interleaving pieces, i.e., fault localization and repair synthesis. In fault localization, we propose a metric to quantify the significance of repairing a neuron. We refine the neuron range of neurons and perform linear relaxation on the network within each subinterval. Based on the metric and approximate results, we can identify the neurons that will be repaired. The results of linear relaxation in the previous stage can provide guidance for neuron repair so that neurons are repaired and behave normally.

The overall algorithm of VeRe is shown in Alg. 1. We can see that VeRe works in an iterative way. In each round of repair (an iteration of the while loop in Line 3), we fix exactly one neuron by fault localization and repair synthesis (Line 7–8). After that, we evaluate the new DNN with the available data and record the best model at this stage in \( f^* \), according to both SR/VR and accuracy (Line 11–12). The procedure terminates when all the misclassified data are successfully repaired, outputting this successfully repaired model (Line 13). Note that VeRe only repairs one neuron after a round of fault localization. This is to accommodate the capacity and scalability of existing DNN verification methods as the underlying repair engine of VeRe.

#### 3.1 Fault Localization

While deep neural networks have a large number of neurons, the malicious behaviours of a DNN are often dominated by a relatively small class of neurons [15, 29, 48, 73]. Therefore, VeRe starts with localizing these faulty neurons to be repaired.

We assume that, for neuron \( j \) in layer \( i \) of a DNN \( f \), its behavior is expected to be in an interval \( I_{ij} \), which we call the neuron range of \( j \). In practice, the interval \( I_{ij} \) can be obtained according to its behaviors on the samples, i.e., \( [\min_{x \in D} h_j(x), \max_{x \in D} h_j(x)] \). Since the samples are limited, it should be suspected whether this interval is representative. Here we use a parameter \( \kappa \geq 1 \) to scale this interval as \( I_{ij} = [\kappa \cdot \max_{x \in D} h_j(x), \kappa \cdot \max_{x \in D} |h_j(x)|] \). The interval \( I_{ij} \) is calculated at the beginning of each iteration, as shown in Line 5–6 of Alg. 1. Here, we allow \( I_{ij} \) to cover the values out of the range of the activation \( a \), e.g., negative values which exceeds the range of ReLU. This will bring us more freedom for repairing the model, so that a better repair effect may be achieved.

In this work, we perform fault localization by quantifying the benefits of repairing a neuron. Specifically, we define a new metric named Repair Significance for this purpose.

**Definition 1 (Repair Significance).** Given a neural network \( f \) and a misclassified sample \( x \), the Repair Significance for the neuron \( j \) in layer \( i \) with its neuron range \( I_{ij} \) is

\[
R_{i,j}(x) = \max_{v \in I_{ij}} [f'_2(h'(x) | j \leftarrow v) - f'_2(h'(x))], \tag{3}
\]
Algorithm 2 Fault localization

Input: DNN \( f \), set \( D_m \) of misclassified inputs, and neuron ranges \( I_{ij} \) for all the neurons \( j \) in layer \( i \).

Output: The candidate neuron index \( j^* \) and the lower bound \( g_{i,j^*} \).

1. function \textsc{FaultLocalization}(f, \( i, D_m, (I_{ij})_j \))
2. for \( x \in D_m \) do
3. for neuron \( j \) in layer \( i \) do
4. Split \( I_{ij} \) into \( K \) intervals \( (I_{ij,k})_{k=1}^K \) evenly
5. for \( k \leftarrow 1 \) to \( K \) do
6. \( g_{i,j,k} \leftarrow \text{CROWN}(f_{ik}^j, I_{ij,k}) \)
7. \( g_{ij} \leftarrow \sum_{k=1}^K g_{i,j,k} \cdot 1_{I_{ij,k}} \) \( \triangleright \text{I}_j^f \): indicator function on \( I \)
8. \( \tilde{R}_{ij}(x) \leftarrow \max_{k=1...K} g_{ij}(v) - f^j_k(h^j(x)) \)
9. \( \tilde{R}_{ij}(x) \leftarrow \frac{\tilde{R}_{ij}(x)}{\text{Normalization}} \)
10. for neuron \( j \) in layer \( i \) do
11. \( \hat{R}_{ij} \leftarrow \sum_{x \in D_m} \tilde{R}_{ij}(x) \)
12. \( j^* \leftarrow \text{arg max}_j \hat{R}_{ij} \)
13. return \( j^*, g_{i,j^*} \)

where \( f_{ik}^j \) is the difference between the scores of the correct classification label and its output classification label w.r.t. \( x \) in the output of \( f^j \), and \( h^j(x)[j \leftarrow v] \) is obtained from \( h^j(x) \) by substituting the \( j \)th entry of \( h^j(x) \) with the real value \( v \).

Given a misclassified sample \( x \), we have \( f_{ik}^j(h^j(x)) < 0 \). If there exists \( v \in I_{ij} \) such that \( f_{ik}^j(h^j(x)[j \leftarrow v]) > 0 \) is positive, the misclassified input can be correctly classified after the neuron is repaired. Intuitively, \( R_{ij}(x) \) measures the maximum effect that can be achieved on correctly classifying \( x \) if the best patching is conducted on neuron \( j \). For the misclassified sample set \( D_m \), the corresponding Repair Significance can be calculated by summing \( R_{ij}(x) \) of each sample, i.e., \( R_{ij} = \sum_{x \in D_m} R_{ij}(x) \).

Note that solving \( v \) for a given sample \( x \) to maximize \( R_{ij}(x) \) in Eq. (3) is a non-convex optimization problem, which is hard to solve. To obtain an estimation of \( R_{ij}(x) \), we employ the verification tool CROWN to conduct a linear approximation to \( f_{ik}^j(h^j(x)[j \leftarrow v]) \) in Eq. (3) for \( v \in I_{ij} \). Specifically, we perform linear relaxation and bound propagation on the sub-network \( f^j \). Unlike general verification tasks, we apply perturbations (i.e., repairs) to hidden neurons instead of input neurons. Given the sub-network \( f^j \) and an input \( h^j(x) \) with the perturbation \( v \in I_{ij} \) on its \( j \)th coordinate, we perform linear relaxation with CROWN. By propagating the upper and lower bounds layer by layer, we finally obtain a lower bound of \( f^j \) in the form of \( g_{ij}(v) = av + b, \) which satisfies
\[ \forall v \in I_{ij}, g_{ij}(v) \leq f_{ik}^j(h^j(x)[j \leftarrow v]). \]

Note that the sub-network \( f^j \) may still be a multi-layer neural network. Namely, it is likely to be highly non-linear, which leads to a linear relaxation of low precision. To reduce the low precision in linear relaxation, we further split the neuron range evenly into \( K \) intervals as \( I_{ij,k} = \bigcup_{j=1}^K I_{ij,k} \), and the linear relaxation is applied to each of these intervals. Then, the lower bound \( g_{ij} \) in the form of a piecewise-linear function, affine on each \( I_{ij,k} \), is obtained, with which the Repair Significance can be estimated more accurately as
\[ \hat{R}_{ij}(x) = \max_{k=1...K} g_{ij}(v) - f_{ik}^j(h^j(x)). \] (4)

We emphasize that, formal verification is a suitable way to efficiently obtain a sound estimation of \( R_{ij}(x) \), i.e., \( \hat{R}_{ij}(x) \leq R_{ij}(x) \), and their difference converges to 0 as \( K \to \infty \). As a sound estimation, \( \hat{R}_{ij}(x) \) measures the Repair Significance in a conservative manner, which would contribute to high stability in the repair effectiveness.

Alg. 2 shows the details of the fault localization phase. The procedure is consistent with what we have described above. In particular, considering that all the misclassified samples enjoy a coordinate position, we add a normalization after we obtain \( R_{ij}(x) \) for all the neurons \( j \). Namely, for each \( x \in D_m \), the \( \hat{R}_{ij}(x) \) of the neuron that achieves the maximum Repair Significance will be mapped to 1, and the others are linearly scaled (Line 9). The estimation of the total Repair Significance for neuron \( j \) is the sum of \( \hat{R}_{ij}(x) \) over \( x \in D_m \), and we find the neuron with the largest total Repair Significance estimation as the candidate neuron, on which repair will be synthesized in this iteration. We also save the linear relaxation \( g_{ij} \) as the guidance of repair synthesis.

3.2 Repair Synthesis

Next, we present how VaRe repairs the candidate neuron \( j^* \) in an iteration. Given a misclassified sample \( x \) and a candidate neuron \( j^* \), there are two challenges to be addressed:

(C1) How to find the ideal interval for \( j^* \) to repair a misclassified sample \( x \). An ideal interval should be able to effectively alleviate the wrong behavior of the DNN on \( x \) without affecting the original performance of the network.

(C2) How to modify the weights on the neuron \( j^* \) so that its activation value on \( x \) lies within the ideal interval.

To address (C1), we utilize the results of fault localization to infer the target activation value for the candidate neuron. For a misclassified sample \( x \in D_m \), a target interval \( s_{ij^*}(x) = (I_{ij^*}, k)_{k=1}^K \) is ideal, if it satisfies the following two conditions:

(1) The linear lower bound \( g_{i,j^*} \) is always positive on \( s_{ij^*}(x) \).
(2) The activation value after repair is the closest to the original.

The first condition implies that the error must be successfully removed, if the neuron \( j^* \) behaves within the ideal interval \( s_{ij^*}(x) \). The second condition is to minimize the change in the value of the neuron so that the original performance of the network can be mostly preserved. Denote by \( d(a, I) = \min_{a \in I} |r - a| \) the distance between \( r \in \mathbb{R} \) and an interval \( I \). Then, the ideal interval \( s_{ij^*}(x) \) can be assigned as any element in
\[ \arg \min_{I_{ij^*}, k \in T} d(h_{ij^*}(x), I_{ij^*}, k). \] (5)

where \( T = \{ I_{ij^*, k} \mid \forall v \in I_{ij^*, k} \text{ and } g_{i,j^*}(v) > 0 \} \) is the set of all the intervals satisfying the condition (1). Fig. 2(a) shows intuitively where the ideal interval is. Additionally, a situation may occur that

Figure 2: The target interval in the two cases.

\[ \text{original} \quad \text{ideal interval} \quad \text{target} \]
Algorithm 3 Repair synthesis

Input: DNN $f$, set of inputs including the misclassified inputs and the correctly classified inputs $D = D_m \cup D_c$, candidate neuron $j^*$ in layer $i$, and the linear lower bound $g_{i,j^*}$ on $I_{i,j^*}$.

Output: The repaired DNN $f'$ as the repair synthesis for $f^*$

1: function RepairSynthesis($f', D_m, D_c, j^*, g_{i,j^*}$)
2:   for $x \in D_m$ do
3:       Ideal $\leftarrow$ False, $T \leftarrow \emptyset$
4:   for $k \leftarrow 1$ to $K$ do
5:       if $\min_{e \in I_{i,j^*_k}} g_{i,j^*_k}(v) > 0$ then
6:           Ideal $\leftarrow$ True, $T \leftarrow T \cup \{I_{i,j^*_k}\}$
7:   if Ideal $=$ True then
8:       $k^* \leftarrow \arg \min_{k \in \mathbb{N}^{\geq 1}} I_{i,j^*_k} \in T$ (h($x$), $I_{i,j^*_k}$)
9:       $s_{i,j^*_k}^*(x) \leftarrow I_{i,j^*_k}$
10:   else
11:       $o^* \leftarrow \arg \max_{e \in I_{i,j^*_k}} g_{i,j^*_k}(v)$
12:       $s_{i,j^*_k}^*(x) \leftarrow [o^*, o^*]$
13:   end
14: end
15: $L \leftarrow L_c + L_m$ according to Eq. (5) and Eq. (6)
16: $\hat{w}_{i,j^*} \leftarrow w_{i,j^*}, \hat{b}_{i,j^*} \leftarrow b_{i,j^*}, \alpha \leftarrow 1, \beta \leftarrow 0$ → Initialization
17: $(\nu_{i,j^*}', \hat{h}_{i,j^*}') \leftarrow \min\{\nu_{i,j^*}, \hat{h}_{i,j^*}, \alpha; \beta; \min L\}$
18: $w_{i,j^*} \leftarrow w_{i,j^*}', \hat{b}_{i,j^*} \leftarrow \hat{h}_{i,j^*}'$ → In-weights
19: $b_{f',i} + \nu_{f',i} = \nu_{f',i}' \leftarrow b_{f',i} + \nu_{f',i}', \nu_{f',i} = \alpha \nu_{f',i}' \leftarrow \alpha \nu_{f',i} \leftarrow \nu_{f',i}' \leftarrow \nu_{f',i}'$ → Out-weights
20: return the current DNN $f'$

the condition (1) does not hold for any $I_{i,j^*_k}$. In this case, we simply choose the endpoint (of some interval $I_{i,j^*_k}$) that produces the largest $g_{i,j^*}(v)$ as the target, as shown in Fig. 2(b). For consistency, we consider the target value to be a single point interval $s_{i,j^*_k}^*(x) = [o^*, o^*]$, where $o^* = \arg \max_{e \in I_{i,j^*_k}} g_{i,j^*_k}(v)$. In Line 2–12 of Alg. 3, we show the procedure of calculating the target interval $s_{i,j^*_k}^*(x)$, where a Boolean variable “Ideal” marks whether the condition (1) holds. The interval with the largest $g_{i,j^*}(v)$ may not be the ideal interval. Therefore, we propose two strategies for selecting intervals: If we want the best repair effects, we should choose the interval with the maximum $g_{i,j^*}(v)$; if we want to balance the preservation of the model’s original performance and the repair effects, we can choose the ideal interval. In this work, we select the ideal interval to better protect model performance in the backdoor removal scenario, and the interval with the maximum $g_{i,j^*}(v)$ for safety property violation repair.

Here the linear lower bound $g_{i,j^*}$ obtained by formal verification again helps extract the target interval of the neuron $j^*$ for every misclassified sample. Due to the soundness of formal verification, for a certain input $x \in D_m$, it is sufficient to infer its correct label with its ideal interval of the neuron $j^*$ (if the ideal interval exists). Thus, we intend to adjust the weights associated with the neuron $j^*$ so that for as many samples $x \in D_m$ as possible, the values of the neuron will fall within their ideal intervals, respectively. This is exactly what (C2) does.

To address (C2), we consider which weights on the neuron $j^*$ are to be modified. For a poisoned backdoor model, the behavior of a candidate neuron on a misclassified sample may include both the backdoored behavior and the function for correctly classifying a certain class. This phenomenon inspires us that, neuron-level patches like scaling or adding a bias to the value of the candidate neuron, which is popular among existing methods like [67], may not effectively remove all backdoor behaviors while preserving original performance of the model. Therefore, we replace the candidate neuron $j^*$ by modifying its in-weights and out-weights. We use a mini-network $f_{mini}$ to substitute the candidate neuron $j^*$, including its in-weights and out-weights, in the current DNN. It contains the same number of parameters $w_{i,j^*}$ and $b_{i,j^*}$ as the in-weights and the bias of $j^*$, and two extra parameters $\alpha$ and $\beta$ for out-weight modification. It receives the output of the $(i-1)$th layer $h^{i-1}$ as input, and outputs the repaired behavior of $j^*$ as

$$f_{mini}(h^{i-1}) = \alpha \cdot \sigma(w_{i,j^*}^T \cdot h^{i-1} + b_{i,j^*}) + \beta.$$ (5)

We remark that, through the additional linear transformation with parameters $\alpha$ and $\beta$, the output of this mini-network can exceed the range of the activation function $\sigma$, which can solve the problem that the ideal interval may be outside the output range of the activation function. The mini-network $f_{mini}$, as the repair of $j^*$, will not change the original structure of the DNN, because we can construct an equivalent DNN without change in structure. Specifically, the out-weights of $j^*$ are all scaled with $\alpha$, and the biases are shifted with the multiplication of $\beta$ and the original corresponding out-weight, as shown in Line 16–17 of Alg. 3.

Next, we design a loss function as $L = L_c + L_m$ to optimize the weights of mini-network $f_{mini}$, where

$$L_c = \frac{1}{|D_c|} \sum_{x \in D_c} (f_{mini}(h^{i-1}(x)) - h^{i-1}(x))^2,$$

$$L_m = \frac{1}{|D_m|} \sum_{x \in D_m} (d(f_{mini}(h^{i-1}(x)), s_{i,j^*_k}^*(x)))^2.$$ (6)

Intuitively, $L_c$ enforces the output of $f_{mini}$ for the correctly classified samples to be similar to the output of the original neuron. For those misclassified, we use $L_m$ to guide the output of $f_{mini}$ to move towards the ideal intervals. Specifically, when the output of $f_{mini}$ for $h^{i-1}(x)$ is already within the ideal interval, the corresponding loss function is 0, so that no further changes for repair $x$ are made to $f_{mini}$. Unlike the typical loss function that focuses on the output of the whole DNN, $L_c$ directly measures the distance between the output of $f_{mini}$ and the ideal interval. Thus enabling a more effective correction of its erroneous behaviors and the candidate neuron will not ‘over-learn’ samples that have been correctly classified, avoiding overfitting and effectively correcting incorrect behavior. We use the gradient descent algorithm to optimize the weight of $f_{mini}$. Specifically, we choose Adam [41] as the optimizer.

To further improve the efficiency, we divide the available misclassified data into several batches. In each round, we randomly select a batch of data to perform fault localization and neuron repair.

## 4 EVALUATION

In this section, we conduct a set of experiments to evaluate VeRe. We report the experiment results for answering the following five research questions.

**RQ1**: Can VeRe repair a DNN more effectively and efficiently compared with the state-of-the-art?

**RQ2**: How does the number of samples and iterations influence the performance of VeRe?

**RQ3**: What role does interval splitting play in VeRe?
636

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Model</th>
<th>Train</th>
<th>Repair</th>
<th>Generalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>CNN</td>
<td>60,000</td>
<td>clean</td>
<td>poisoned</td>
</tr>
<tr>
<td>CIFAR10</td>
<td>VGG13</td>
<td>50,000</td>
<td>1,000</td>
<td>1,000</td>
</tr>
<tr>
<td>SVHN</td>
<td>VGG13</td>
<td>73,257</td>
<td>1,000</td>
<td>1,000</td>
</tr>
<tr>
<td>GTSRB</td>
<td>VGG11</td>
<td>39,200</td>
<td>1,000</td>
<td>1,000</td>
</tr>
<tr>
<td>ImageNet</td>
<td>VGG14</td>
<td>9,406</td>
<td>200</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 1: Number of the records in the datasets.

RQ4: How coupled is the repair process with the specific localisation and vice versa?

RQ5: Is VeRe scalable to high dimensional input and DNNs with other activation functions?

4.1 Experiment Setup

We apply VeRe to two repair tasks: 1) removing backdoor and 2) correcting safety property violation. In total, we evaluate VeRe with 5 baselines, 2 backdoor attack methods, and 46 models across 6 datasets. We run all the experiments 5 times and report the mean results.

4.1.1 Removing backdoor. Two popular backdoor attacks, BadNets [29] and Blend [16], are used in the experiment. For BadNets attack, a random noise square measuring 5 × 5 pixels is placed in the lower right corner of the image as the trigger. For Blend attack, we generate a trigger pattern by sampling pixel values from a uniform distribution in the range [0, 255], and then attach the trigger t to the sample x according to the injection strategy of Blend, i.e., 

\[ t \cdot (1 - y) \cdot x, \text{ where we set the ratio } y = 0.2. \]

There are five datasets: MNIST [43], CIFAR-10 [42], SVHN [54], GTSRB [66] and ImageNet [53]. The original training sets are only used to train the poisoned neural networks. We divide the original test set into two parts: the Repair set D_r and the Generalization set D_g. Subsequently, we inject malicious triggers into the repair set and the generalization set to generate the poisoned sets D_r and D_g, respectively. We randomly select 1,000 clean samples and 1,000 poisoned samples from D_r and D_g as the available data, respectively. We use D_r and D_g to evaluate model’s generalization ability. ImageNet is a subset of ImageNet that consists of ten categories. Due to the relatively small size of the dataset, we establish slightly smaller repair set and test sets. The numbers of data in these sets, as well as the DNN models and their architectures for each dataset, are shown in Table 1.

4.1.2 Correcting safety property violation. We evaluate VeRe over 36 ACAS Xu [36, 37] networks. Each network in ACAS Xu consists of six hidden layers, each with 50 ReLU neurons. As reported in [20], 34 models violate Property-2, 1 model violates Property-7, and 1 model violates Property-8, resulting in 36 repairing tasks.

We sample 10,000 non-violating samples and 10,000 counterexamples as the Repair set D_r, and use independent 10,000 counterexamples as the Generalization set D_g. We utilize a drawdown set to assess the extent to which the original performance of the repaired model is affected. Specifically, we select 3 properties (including the property to be repaired) for each model, and sample 5,000 non-violating instances from the state space of each property. Finally, we generate a drawdown set of size 15,000 for each model.

4.1.3 Baselines and metrics. We implement and compare 3 state-of-the-art (SOTA) methods with our method to evaluate their performance on backdoor removal, including AI-Lancet [89], CARE [67] and RM [32]. We configure each baseline according to the best performance settings reported in its respective paper. Specifically, AI-Lancet proposed an optimization method for trigger restoration to obtain poisoned samples. In order to ensure fairness, we skip this step and provide real triggers directly. We formulate the accuracy (Acc) and the attack success rate (ASR) of a DNN under backdoor attack as follows:

\[ \text{Acc} = \frac{\sum_{x \in D_r} [f_r(x) = y_t]}{|D_r|} \quad \text{and} \quad \text{ASR} = \frac{\sum_{x \in D_r} [f_r(x) = t]}{|D_r|}, \]

where \( y_t \) is the true label of \( x \), \( t \) is the target label of the backdoor, and \([\cdot]\) is the Iverson bracket that takes the value 1 if the statement is true and 0 otherwise. We further define the defense success rate (DSR) for repairing a backdoored model as \( \text{DSR} = 1 - \text{ASR} \). Note that the Acc and the DSR are evaluated on a Generalization set.

For the task of fixing safety property violation, we compare VeRe with CARE, PRDNN and REASSURE [24]. For a repaired model \( f' \), the repair success rate (RSR), generalization and the drawdown can be computed as follows:

\[ \text{RSR} = \frac{\sum_{x \in D_r} [f_r(x) \in P]}{|D_r|} \quad \text{Generalization} = \frac{\sum_{x \in D_g} [f_r(x) \in P]}{|D_g|}, \]

\[ \text{Drawdown} = \frac{1}{\Psi} \sum_{x \in D_r} \frac{\sum_{e \in \Psi} [f_r(x) \in P_e]}{|D_r|}. \]

where \( P \) is the output set of the safety property to be repaired. The set \( \Psi \) represents the properties we select for evaluating drawdown.

4.2 Comparison with Baselines

Correcting Safety Property Violation. We first compare VeRe with CARE and PRDNN for correcting the violation of safety properties, and the results are shown in Table 2. PRDNN constructs a provable repair, thereby achieving a 100% RSR on the Repair set. VeRe also achieves high RSR (≥ 99.8%), while CARE achieves an average RSR of 94.89%. In terms of generalization, all tools show impressive performance, while VeRe achieves the best generalization, at 99.87%. For 28 out of 36 models, VeRe achieves 100% generalization, which means that after repair, the models satisfy their safety properties on all original counterexamples in the generalization sets. As a comparison, CARE and PRDNN have an average generalization of 94.70% and 95.15%, respectively. Additionally, VeRe demonstrate better performance on the drawdown set, with an average drawdown of 15.26% for the 36 repaired models, whereas CARE and PRDNN display drawdown of 19.53% and 19.48% respectively.

For this repairing task, both PRDNN and our method demonstrate a significant efficiency advantage. PRDNN converts the repairing task to a linear programming problem and thus achieves the least time cost. VeRe is capable of repairing most of the models within 6 seconds, which is 20 times faster than CARE on average. REASSURE is another provable repair baseline. Due to its significant time cost, we cannot directly add it to our experiments with exactly the same setup. Thus we set 4 scenarios with fewer samples.

REASSURE’s generalization increases as the number of samples increases, while VeRe is always better. REASSURE achieves better
Table 2: Results of repairing violation of safety properties

<table>
<thead>
<tr>
<th>Attack</th>
<th>Dataset</th>
<th>Before</th>
<th>CARE</th>
<th>Al-Lancet</th>
<th>RM</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Acc</td>
<td>ASR</td>
<td>DSR</td>
<td>Time</td>
<td>Acc</td>
</tr>
<tr>
<td>BadNets</td>
<td>MNIST</td>
<td>97.96</td>
<td>99.01</td>
<td>99.68</td>
<td>98.84</td>
<td>99.98</td>
</tr>
<tr>
<td></td>
<td>SVHN</td>
<td>97.93</td>
<td>99.95</td>
<td>84.94</td>
<td>99.66</td>
<td>709.32</td>
</tr>
<tr>
<td></td>
<td>CIFAR-10</td>
<td>92.82</td>
<td>99.97</td>
<td>77.48</td>
<td>98.40</td>
<td>920.92</td>
</tr>
<tr>
<td></td>
<td>GTSRB</td>
<td>90.07</td>
<td>98.91</td>
<td>58.56</td>
<td>90.82</td>
<td>558.58</td>
</tr>
<tr>
<td></td>
<td>AVG</td>
<td>91.58</td>
<td>99.46</td>
<td>80.18</td>
<td>99.72</td>
<td>645.41</td>
</tr>
<tr>
<td>Blend</td>
<td>MNIST</td>
<td>97.95</td>
<td>99.97</td>
<td>92.80</td>
<td>99.93</td>
<td>990.24</td>
</tr>
<tr>
<td></td>
<td>SVHN</td>
<td>97.91</td>
<td>99.94</td>
<td>83.08</td>
<td>98.31</td>
<td>2041.70</td>
</tr>
<tr>
<td></td>
<td>CIFAR-10</td>
<td>94.97</td>
<td>99.43</td>
<td>69.03</td>
<td>99.74</td>
<td>728.17</td>
</tr>
<tr>
<td></td>
<td>GTSRB</td>
<td>91.35</td>
<td>98.79</td>
<td>62.48</td>
<td>99.56</td>
<td>2017.71</td>
</tr>
<tr>
<td></td>
<td>AVG</td>
<td>91.60</td>
<td>97.48</td>
<td>76.85</td>
<td>95.63</td>
<td>1444.46</td>
</tr>
</tbody>
</table>

Table 3: Results of backdoor removal, where we mark the overall best values \textit{bold}.

<table>
<thead>
<tr>
<th>Attack</th>
<th>Dataset</th>
<th>Before</th>
<th>CARE</th>
<th>Al-Lancet</th>
<th>RM</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Acc</td>
<td>ASR</td>
<td>DSR</td>
<td>Time</td>
<td>Acc</td>
</tr>
<tr>
<td>BadNets</td>
<td>MNIST</td>
<td>97.92</td>
<td>99.02</td>
<td>99.68</td>
<td>98.84</td>
<td>99.98</td>
</tr>
<tr>
<td></td>
<td>SVHN</td>
<td>97.92</td>
<td>99.95</td>
<td>84.94</td>
<td>99.66</td>
<td>709.32</td>
</tr>
<tr>
<td></td>
<td>CIFAR-10</td>
<td>92.82</td>
<td>99.97</td>
<td>77.48</td>
<td>98.40</td>
<td>920.92</td>
</tr>
<tr>
<td></td>
<td>GTSRB</td>
<td>90.07</td>
<td>98.91</td>
<td>58.56</td>
<td>90.82</td>
<td>558.58</td>
</tr>
<tr>
<td>Blend</td>
<td>MNIST</td>
<td>97.95</td>
<td>99.97</td>
<td>92.80</td>
<td>99.93</td>
<td>990.24</td>
</tr>
<tr>
<td></td>
<td>SVHN</td>
<td>97.91</td>
<td>99.94</td>
<td>83.08</td>
<td>98.31</td>
<td>2041.70</td>
</tr>
<tr>
<td></td>
<td>CIFAR-10</td>
<td>94.97</td>
<td>99.43</td>
<td>69.03</td>
<td>99.74</td>
<td>728.17</td>
</tr>
<tr>
<td></td>
<td>GTSRB</td>
<td>91.35</td>
<td>98.79</td>
<td>62.48</td>
<td>99.56</td>
<td>2017.71</td>
</tr>
</tbody>
</table>

4.3 Monotonicity w.r.t. the number of samples and rounds

In this section, we study how \textit{VeRe} performs on repairing DNNs with different number of available samples and how does the repair effect change after each round of repair.

For safety violation repairing, we consider four different sample size configurations: 500 positive samples and \{500, 200, 100, 50\} negative samples, respectively. We evaluate the performance of \textit{VeRe} under varying amounts of available samples, and the experimental
results are presented in Table 4. VeRe effectively repairs models even with limited samples. Specifically, with access to 500 counterexamples, the minimum RSR of our method is 99.75% and 23 models achieve 100% generalization. We further reduce the number of available negative samples. The average RSR of the repaired models exceeds 98% in all scenarios, with the largest decrease being 2.78%. Furthermore, VeRe demonstrates stable repair ability in scenarios with limited data, with the DSR of the repaired models being above 95% except on GTSRB with only 50 available samples. The results in Table 5 confirm that VeRe indeed makes more efficient use of poisoned samples by symbolic analysis. When the number of poisoned samples is reduced from 1 000 to 50, the repair effect of VeRe only decreases by 1% to 7%, demonstrating the superior effectiveness of VeRe in data scarcity scenarios compared to baselines.

To investigate how our method performs after each round of repair, we record the performance of the model (including generalization) after each round. The experimental results are shown in the Fig. 3. We find that DSR can be improved to varying degrees after each round of repair, with negligible effects on the original performance, which also shows that the neurons and intervals we locate are indeed useful.

**Answer to RQ2:** Even in situations where data availability is restricted, VeRe can still efficiently and effectively remove erroneous behaviors while preserving models’ original performance. Compared to alternative approaches, VeRe can make better use of the available data by symbolic analysis.

### 4.4 Effect of Interval Splitting

Recall that we split the neuron range $I_i$ into $K$ disjoint sub-intervals to reduce the approximation error. In this experiment, we study the effect of interval splitting. We maintain the same settings as section 4.2 and record the changes in the Acc and DSR of the repaired model under scenarios with different number of subintervals. The results are shown in the Fig. 4. We find that more fine-splited interval can better preserve the original performance of the model on the CIFAR-10 and SVHN dataset under badnets attacks. Compared to not splitting intervals, dividing the intervals can bring an accuracy improvement of 1.67% to 8.23%.

For the SVHN dataset under blend attacks, dividing the intervals does not bring significant accuracy improvement. Therefore, we further investigate whether the repaired neurons play an important role in correctly classifying a certain sample category. We record
we compare the classification performance of the frozen model. As a comparison, the loss function in our repair method does not consist of two steps, i.e., fault localization and repair synthesis. In this section, we study how the coupling between localization step and the repair step. We replace our repair process with other repair methods. Specifically, we use particle swarm optimization method from CARE, RM’s fine-tune and optimization with regularization[30] as three baselines. We denote the replaced baselines as PSO*, RM*, and Reg* respectively. We show the results in Table 6. Kindly refer to [1] for more detailed results.

4.5 The coupling between the two steps
VeRe consists of two steps, i.e., fault localization and repair synthesis. In this section, we study how the coupling between localization step and the repair step. We replace our repair process with other repair methods. Specifically, we use particle swarm optimization method from CARE, RM’s fine-tune and optimization with regularization[30] as three baselines. We denote the replaced baselines as PSO*, RM*, and Reg* respectively. We show the results in Table 6. Kindly refer to [1] for more detailed results.

Under the BadNets attack, most baselines can protect the original performance of the model except for the PSO*. Among them, Reg* performs best, with an average Acc drop of 1.07%. Our method also protects the model performance well, with a drop of 1.54%. For improving DSR, VeRe performs best, with a repaired average DSR of 99.74%. In comparison, the best-performing RM* in the baselines can only improve DSR to 96.06%. Reg* overfits severely when the number of available samples is limited (even if regularization is used to prevent overfitting), and it obtains 99.70% Acc and 100.00% DSR on the repair set but performs poorly on the generalization set.

As a comparison, the loss function in our repair method does not promote the candidate neuron to over-learn samples that have been correctly classified, avoiding overfitting and effectively correcting erroneous behavior. For the Blend attack, no baseline can increase DSR while maintaining Acc. PSO* improve DSR to nearly 100.00%, but Acc drops severely (more than 20%). RM* and Reg* suffer from catastrophic forgetting and overfitting, respectively.

In addition, we find that compared to RM, RM* can more effectively increase DSR while maintaining the same level of Acc. This indicates that the verification-based fault localization has found more appropriate candidate neuron for repair. The effect of PSO* is at the same level as CARE: it cannot protect the original performance of the model and cannot effectively remove backdoor in...
To investigate the scalability of VeRe, we replace the ReLU units in each neural network 1.11%. The average time overhead of to [1] for detailed results. achieve an average DSR of 96.59%, while Acc decreases by up to 3.25%. Under Blend attack, the performance of AI-Lancet in this scenario, as it requires running multiple experiments. Unfortunately, we are unable to reproduce in the table to represent the average results from multiple experi-

Note that due to the unstable performance of CARE, we use “CARE”

some scenarios, which indicates that simple combination of our localization method with other repair methods may not work well.

**Answer to RQ4:** Our fault localization step and repair step are highly coupled. Compared to simply combining our localization method with other repair methods, VeRe remove erroneous behaviors more effectively while protecting the performance of the model. In addition, our verification-based localization method locate more accurate than gradient-based method.

### 4.6 Scalability

To investigate the scalability of VeRe we conduct a backdoor re-

The results are presented in the Table 7. Note that due to the unstable performance of CARE, we use “CARE” in the table to represent the average results from multiple experiments, while “CARE” represents the best performance observed in multiple experiments. Unfortunately, we are unable to reproduce the performance of Al-Lancet in this scenario, as it requires running three network models simultaneously, resulting in out of memory.

According to the Table 7, CARE is unable to repair the network under BadNets attack, and the DSR after repair is 2.30%. The DSR of RM is 84.27%, while the model accuracy decreases by 8.93%. In comparison, our method can increase the DSR to over 98% while slightly sacrificing accuracy (less than 3%). Under Blend attack, CARE improves DSR to 100.00% (only one in ten experiments), but suffers from catastrophic forgetting. In addition, the time over-

To study whether VeRe generalizes to networks with other acti-

The experimental results show that VeRe are quite different, which provides guidance for fault localization and target intervals obtained by CROWN in VeRe directly and significantly reflects the repair significance of each neuron and how to repair a candidate neuron, and such guidance is conservative due to the soundness of formal verification, which is beneficial for the stability of the repair effects. In the future, we plan to incorporate more program synthesis techniques [18] into VeRe for repairing DNNs.

**DNN verification.** In 2010, the first DNN verification algorithm based on partition refinement was proposed in [57]. In the past decade, numerous formal verification techniques have been proposed for verifying DNNs, primarily including constraint solving [10, 22, 27, 35, 39, 40, 46, 53], abstract interpretation [26, 44, 61–

### 6 CONCLUSION

We propose VeRe, a novel verification guided synthesis framework for repairing DNNs. VeRe performs linear relaxation on fully con-

The experimental results show that VeRe can repair various models efficiently and effectively, while preserving original performance of the model. We have to claim that VeRe still has some limitations. Since it relies on a verification engine for fault localization and repair synthesis, VeRe only repairs the properties that can be formally specified, and currently it cannot repair violation of fairness. The weight modification in the repair synthesis only works for fully-connected layers, and we lack a repair strategy for more structures like convolutional layers. As for future works, we are eager to explore how VeRe is used for fairness repair, and design the repair strategy for convolutional layers.