VeRe: Verification Guided Synthesis for Repairing Deep Neural Networks

Jianan Ma
Hangzhou Dianzi University
Zhejiang University
Hangzhou, China
mjnn@hdu.edu.cn

Pengfei Yang
SKLCS, Institute of Software, CAS
Beijing, China
yangpf@ios.ac.cn

Jingyi Wang*
Zhejiang University
ZJU-Hangzhou Global Scientific and Technological Innovation Center
Hangzhou, China
wangjyee@zju.edu.cn

Youcheng Sun
University of Manchester
Manchester, United Kingdom
youcheng.sun@manchester.ac.uk

Cheng-Chao Huang
Nanjing Institute of Software Technology, ISCAS
Nanjing, China
chengchao@nj.iscas.ac.cn

Zhen Wang
Hangzhou Dianzi University
Hangzhou, China
wangzhen@hdu.edu.cn

ABSTRACT

Neural network repair aims to fix the 'bugs'[1] of neural networks by modifying the model's architecture or parameters. However, due to the data-driven nature of neural networks, it is difficult to explain the relationship between the internal neurons and erroneous behaviors, making further repair challenging. While several work exists to identify responsible neurons based on gradient or causality analysis, their effectiveness heavily rely on the quality of available 'bugged' data and multiple heuristics in layer or neuron selection. In this work, we address the issue utilizing the power of formal verification (in particular for neural networks). Specifically, we propose VeRe, a verification-guided neural network repair framework that performs fault localization based on linear relaxation to symbolically calculate the repair significance of neurons and furthermore optimize the parameters of problematic neurons to repair erroneous behaviors. We evaluated VeRe on various repair tasks, and our experimental results show that VeRe can efficiently and effectively repair all neural networks without degrading the model's performance. For the task of removing backdoors, VeRe successfully reduces attack success rate from 98.47% to 0.38% on average, while causing an average performance drop of 0.9%. For the task of repairing safety properties, VeRe successfully repairs all the 36 tasks and achieves 99.87% generalization on average.

CCS CONCEPTS

• **Security and privacy → Software and application security**;
• **Computing methodologies → Artificial intelligence**.

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

ICSE '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0217-4/24/04…$15.00
https://doi.org/10.1145/3597503.3623332

KEYWORDS

DNN Repair, Verification Guided Synthesis, Fault Localization

ACM Reference Format:
Jianan Ma, Pengfei Yang, Jingyi Wang, Youcheng Sun, Cheng-Chao Huang, and Zhen Wang. 2024. VeRe: Verification Guided Synthesis for Repairing Deep Neural Networks. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3597503.3623332

1 INTRODUCTION

Over the past decade, deep neural networks (DNNs) have brought about breakthroughs in many areas such as computer vision [2], natural language processing [8, 18], 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 [7] 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 [10], adversarial attacks [13, 27], backdoor attacks [47] and distribution drift [68, 82]. To counter these potential risks, periodical 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 [24, 25, 70], input sanitization, fine-tuning, transfer learning [16, 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 [31], 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 data[2]. 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 [6]. Several common verification techniques like [21, 25, 38, 39, 43, 57, 62, 63] have been adapted for DNNs to model or abstract their behaviors and furthermore verify them against safety properties [51] and others [13, 30]. 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 (**Ve**rification Guided Synthesis for **Re**pairing 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.

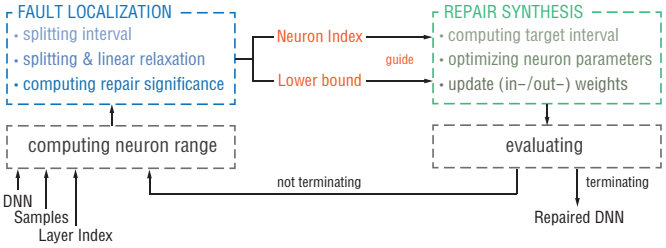

Figure 1: Framework of VeRe.

In summary, we make the following contributions:

- We propose VeRe, a novel verification guided synthesis framework to precisely locate and efficiently fix the erroneous neurons.
- We demonstrate the effectiveness of VeRe in two popular repair tasks, namely violation of safety properties and backdoor attacks.
- We extensively evaluate VeRe on six popular datasets (including Imagenet) and on various neural network architectures. The results show that VeRe significantly outperforms state-of-the-art neuron-level repair methods in terms of effectiveness and efficiency while introducing negligible effect on the model's original performance.
- We release VeRe as an open-source toolkit together with all the experimental datasets [48] to benchmark future research.

2 BACKGROUND

Deep neural network. In this work, we focus on DNNs for classification tasks. A DNN is a function $f : \mathbb{R}^m \to \mathbb{R}^n$ which assigns a high-dimensional input $\boldsymbol{x} \in \mathbb{R}^m$ to an output $f(\boldsymbol{x}) \in \mathbb{R}^n$. A classification DNN chooses the dimension with the highest score as the classification result, i.e., its classification behavior can be described

[2]while VeRe is shown to be effective in presence of both massive or scarce data.

as $C_f(\boldsymbol{x}) = \arg\max_{1\le i\le n} f(\boldsymbol{x})_i$. 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^i to denote the subnetwork from the ith layer (as the input layer) to the output layer of the neural network f, i.e., $f^i = f_{l-1} \circ \cdots \circ f_i$, and h^i to denote the subnetwork from the input layer to the ith layer (as the output layer), i.e., $h^i = f_{i-1} \circ \cdots \circ f_1$. 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 $\boldsymbol{x}$ can be calculated as:

$$h_{ij}(\boldsymbol{x}) = \sigma(\boldsymbol{w}_{ij} \cdot h^{i-1}(\boldsymbol{x}) + b_{ij}) \tag{1}$$

with the activation function σ, the in-weights $\boldsymbol{w}_{ij}$, and the bias b_{ij}.

Safety properties and backdoor attacks. Safety properties refer to those describing 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 $f : \mathbb{R}^m \to \mathbb{R}^n$ be a DNN, $X \subseteq \mathbb{R}^m$ an input set, 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 $\boldsymbol{x} \in X$, $f(\boldsymbol{x}) \in P$. The property violation rate (VR) of a safety property $\varphi = (f, X, P)$ is defined as $\mathrm{VR}(\varphi) = \mathbb{P}(f(\boldsymbol{x}) \notin P \mid \boldsymbol{x} \in X)$.

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 data or model architecture. Similarly, the attack success rate (SR) of a backdoor attack reflects its effectiveness.

Definition 2.2. Let $f : \mathbb{R}^m \to \mathbb{R}^n$ 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 $\mathrm{SR}(f, X) = \mathbb{P}(C_f(\boldsymbol{x}) = t \mid \boldsymbol{x} \in X)$, 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 $z' = \mathrm{ReLU}(z)$ with $z \in [l, u]$ ($l < 0 < u$) can be linearly relaxed to $\frac{uz}{u-l} \le z' \le \frac{u(z-l)}{u-l}$, and the numerical lower bound of z' can be obtained by substituting z in the term $\frac{uz}{u-l}$ 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 $g^{\mathrm{Lower}}(\boldsymbol{x}) = \underline{\boldsymbol{w}}^{\mathrm{T}}\boldsymbol{x} + \underline{\boldsymbol{b}}$ and upper bound $g^{\mathrm{Upper}}(\boldsymbol{x}) = \overline{\boldsymbol{w}}^{\mathrm{T}}\boldsymbol{x} + \overline{\boldsymbol{b}}$ satisfying

$$g^{\mathrm{Lower}}(\boldsymbol{x}) \le f(\boldsymbol{x}) \le g^{\mathrm{Upper}}(\boldsymbol{x}), \forall \boldsymbol{x} \in \Delta. \tag{2}$$

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

> Given a classification DNN $f : \mathbb{R}^m \to \mathbb{R}^n$ 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 removed 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.

Algorithm 1 VeRe

Input: DNN f, set of inputs including the misclassified inputs and the correctly classified inputs $D = D_{\mathrm{m}} \cup D_{\mathrm{c}}$, to-be-repaired layer i, and maximum number of iterations R.
Output: A repaired model f'.

1: **function** VeRe($f, D_{\mathrm{m}}, D_{\mathrm{c}}, i, R$)
2: $f^* \leftarrow f$, $f' \leftarrow f$, iteration $\leftarrow 0$
3: **while** iteration $< R$ **do**
4: iteration $\leftarrow$ iteration + 1
5: **for** neuron j in layer i **do**
6: $\mathcal{I}_{i,j} \leftarrow \left[-\kappa \cdot \max_{\boldsymbol{x}\in D} |h_{ij}(\boldsymbol{x})|, \kappa \cdot \max_{\boldsymbol{x}\in D} |h_{ij}(\boldsymbol{x})|\right]$
▷ h_{ij} is the value of neuron j in layer i in the current f'
7: $j^*,\ g_{i,j^*} \leftarrow$ FaultLocalization($f', i, D_{\mathrm{m}}, (\mathcal{I}_{i,j})_j$) ▷ Alg. 2
8: $f' \leftarrow$ RepairSynthesis($f', D_{\mathrm{m}}, D_{\mathrm{c}}, j^*, g_{i,j^*}$) ▷ Alg. 3
9: **if** $\mathrm{SR}(f', D_{\mathrm{m}}) = 0$ **then**
10: **return** f' ▷ $\mathrm{VR}(\cdot, D)$ instead for safety properties
11: **if** $\mathrm{Acc}(f', D) - \mathrm{SR}(f', D_{\mathrm{m}}) > \mathrm{Acc}(f^*, D) - \mathrm{SR}(f^*, D_{\mathrm{m}})$ **then**
12: $f^* \leftarrow f'$ ▷ $\mathrm{Acc}(f, D)$: the accuracy of f on D
13: **return** f^*

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 f' (Line 9–10), or it reaches a preset number of rounds R and outputs the best model ever f^* (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 [14, 28, 47, 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 $\mathcal{I}_{i,j}$, which we call the neuron range of j. In practice, the interval $\mathcal{I}_{i,j}$ can be obtained according to its behaviors on the samples, i.e., $[\min_{\boldsymbol{x}\in D} h_{ij}(\boldsymbol{x}), \max_{\boldsymbol{x}\in D} h_{ij}(\boldsymbol{x})]$. 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 $\mathcal{I}_{i,j} = \left[-\kappa \cdot \max_{\boldsymbol{x}\in D} |h_{ij}(\boldsymbol{x})|, \kappa \cdot \max_{\boldsymbol{x}\in D} |h_{ij}(\boldsymbol{x})|\right]$. The interval $\mathcal{I}_{i,j}$ is calculated at the beginning of each iteration, as shown in Line 5–6 of Alg. 1. Here, we allow $\mathcal{I}_{i,j}$ to cover the values out of the range of the activation σ, 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 $\boldsymbol{x}$, the Repair Significance for the neuron j in layer i with its neuron range $\mathcal{I}_{i,j}$ is*

$$R_{i,j}(\boldsymbol{x}) = \max_{v\in\mathcal{I}_{i,j}} f^i_{\boldsymbol{x}}(h^i(\boldsymbol{x})[j \leftarrow v]) - f^i_{\boldsymbol{x}}(h^i(\boldsymbol{x})), \tag{3}$$

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

Given a misclassified sample $\boldsymbol{x}$, we have $f^i_{\boldsymbol{x}}(h^i(\boldsymbol{x})) < 0$. Intuitively, $R_{i,j}(\boldsymbol{x})$ measures the maximum effect that can be achieved on fixing $\boldsymbol{x}$ 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 $R_{i,j}(\boldsymbol{x})$ of each sample, i.e., $R_{i,j} = \sum_{\boldsymbol{x}\in D_{\mathrm{m}}} R_{i,j}(\boldsymbol{x})$.

Note that solving v for a given sample $\boldsymbol{x}$ to maximize $R_{i,j}(\boldsymbol{x})$ in Eq. (3) is a non-convex optimization problem, which is hard to solve. To obtain an estimation of $R_{i,j}(\boldsymbol{x})$, we employ the verification tool CROWN to conduct a linear approximation to $f^i_{\boldsymbol{x}}(h^i(\boldsymbol{x})[j \leftarrow v])$ in Eq. (3). Specifically, we perform linear relaxation and bound propagation on the sub-network f^i. Unlike general verification tasks, we apply perturbations (i.e., repairs) to hidden neurons instead of input neurons. Given the sub-network f^i and an input $h^i(\boldsymbol{x})$ with the perturbation $v \in \mathcal{I}_{i,j}$ on its jth 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^i in the form of $g_{i,j}(v) = wv + b$, which satisfies

$$\forall v \in \mathcal{I}_{i,j}\ g_{i,j}(v) \leq f^i_{\boldsymbol{x}}(h^i(\boldsymbol{x})[j \leftarrow v])).$$

Note that the sub-network f^i 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 loss of precision in linear relaxation, we further split the neuron range evenly into K intervals as $\mathcal{I}_{i,j} = \bigcup_{k=1}^{K} \mathcal{I}_{i,j,k}$, and the linear relaxation is applied to each of these intervals. Then, the lower bound $g_{i,j}$ in the form of a piecewise-linear function, affine on each $\mathcal{I}_{i,j,k}$, is obtained, with which the Repair Significance can be estimated more accurately as

$$\widetilde{R}_{i,j}(\boldsymbol{x}) = \max_{\substack{k=1,\ldots,K \\ v\in\mathcal{I}_{i,j,k}}} g_{i,j}(v) - f^i_{\boldsymbol{x}}(h^i(\boldsymbol{x})). \tag{4}$$

We emphasize that, formal verification is a suitable way to efficiently obtain a sound estimation of $R_{i,j}(\boldsymbol{x})$, i.e., $\widetilde{R}_{i,j}(\boldsymbol{x}) \leq R_{i,j}(\boldsymbol{x})$, and their difference converges to 0 as $K \to \infty$. As a sound estimation, $\widetilde{R}_{i,j}(\boldsymbol{x})$ 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 $\widetilde{R}_{i,j}(\boldsymbol{x})$ for all the neurons j. Namely, for each $\boldsymbol{x} \in D_{\mathrm{m}}$, the $\widetilde{R}_{i,j}(\boldsymbol{x})$ 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 $\widetilde{R}_{i,j}(\boldsymbol{x})$ over $\boldsymbol{x} \in D_{\mathrm{m}}$, 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_{i,j^*} as the guidance of repair synthesis.

Algorithm 2 Fault localization

Input: DNN f, set D_{m} of misclassified inputs, and neuron ranges $\mathcal{I}_{i,j}$ for all the neurons j in layer i.
Output: The candidate neuron index j^* and the lower bound g_{i,j^*}.

1: **function** FaultLocalization($f, i, D_{\mathrm{m}}, (\mathcal{I}_{i,j})_j$)
2: **for** $\boldsymbol{x} \in D_{\mathrm{m}}$ **do**
3: **for** neuron j in layer i **do**
4: Split $\mathcal{I}_{i,j}$ into K intervals $(\mathcal{I}_{i,j,k})_{k=1}^{K}$ evenly
5: **for** $k \leftarrow 1$ to K **do**
6: $g_{i,j,k} \leftarrow \mathrm{CROWN}(f^i_{\boldsymbol{x}}, \mathcal{I}_{i,j,k})$
7: $g_{i,j} \leftarrow \sum_{k=1}^{K} g_{i,j,k} \cdot \mathbb{I}_{\mathcal{I}_{i,j,k}}$ ▷ $\mathbb{I}_{\mathcal{I}}$: indicator function on $\mathcal{I}$
8: $\widetilde{R}_{i,j}(\boldsymbol{x}) \leftarrow \max_{\substack{k=1,\ldots,K \\ v\in\mathcal{I}_{i,j,k}}} g_{i,j}(v) - f^i_{\boldsymbol{x}}(h^i(\boldsymbol{x}))$
9: $\widetilde{R}_{i,j}(\boldsymbol{x}) \leftarrow \frac{\widetilde{R}_{i,j}(\boldsymbol{x})}{\max_j \widetilde{R}_{i,j}(\boldsymbol{x})}$ ▷ Normalization
10: **for** neuron j in layer i **do**
11: $\widetilde{R}_{i,j} \leftarrow \sum_{\boldsymbol{x}\in D_{\mathrm{m}}} \widetilde{R}_{i,j}(\boldsymbol{x})$
12: $j^* \leftarrow \arg\max_j \widetilde{R}_{i,j}$
13: **return** j^*, g_{i,j^*}

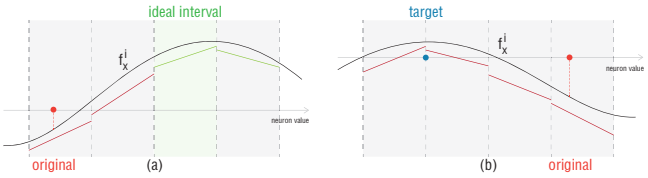

Figure 2: The target interval in the two cases.

Algorithm 3 Repair synthesis

Input: DNN f, set of inputs including the misclassified inputs and the correctly classified inputs $D = D_{\mathrm{m}} \cup D_{\mathrm{c}}$, candidate neuron j^* in layer i, and the linear lower bound g_{i,j^*} on $\mathcal{I}_{i,j^*}$.
Output: The repaired DNN f' as the repair synthesis for j^*

1: **function** RepairSynthesis($f, D_{\mathrm{m}}, D_{\mathrm{c}}, j^*, g_{i,j^*}$)
2: **for** $\boldsymbol{x} \in D_{\mathrm{m}}$ **do**
3: Ideal $\leftarrow$ False, $T \leftarrow \emptyset$
4: **for** $k \leftarrow 1$ to K **do**
5: **if** $\min_{v \in \mathcal{I}_{i,j^*,k}} g_{i,j^*}(v) > 0$ **then**
6: Ideal $\leftarrow$ True, $T \leftarrow T \cup \{\mathcal{I}_{i,j^*,k}\}$
7: **if** Ideal = True **then**
8: $k^* \leftarrow \arg\min_{k:\mathcal{I}_{i,j^*,k} \in T} \mathrm{d}(h_{ij}(\boldsymbol{x}), \mathcal{I}_{i,j^*,k})$
9: $s^*_{i,j^*}(\boldsymbol{x}) \leftarrow \mathcal{I}_{i,j^*,k^*}$
10: **else**
11: $v^* \leftarrow \arg\max_{v \in \mathcal{I}_{i,j^*}} g_{i,j^*}(v)$
12: $s^*_{i,j^*}(\boldsymbol{x}) \leftarrow [v^*, v^*]$
13: $\mathcal{L} \leftarrow \mathcal{L}_{\mathrm{c}} + \mathcal{L}_{\mathrm{m}}$ according to Eq. (5) and Eq. (6)
14: $\tilde{\boldsymbol{w}}_{i,j^*} \leftarrow \boldsymbol{w}_{i,j^*}, \tilde{b}_{i,j^*} \leftarrow b_{i,j^*}, \alpha \leftarrow 1, \beta \leftarrow 0$ ▷ Initialization
15: $(\tilde{\boldsymbol{w}}^*_{i,j^*}, \tilde{b}^*_{i,j^*}, \alpha^*, \beta^*) \leftarrow \mathrm{Adam}(\tilde{\boldsymbol{w}}_{i,j^*}, \tilde{b}_{i,j^*}, \alpha, \beta; \min \mathcal{L})$
16: $\boldsymbol{w}_{i,j^*} \leftarrow \tilde{\boldsymbol{w}}^*_{i,j^*}, b_{i,j^*} \leftarrow \tilde{b}^*_{i,j^*}$ ▷ In-weights
17: $\boldsymbol{b}_{j^*,i+1} \leftarrow \boldsymbol{b}_{j^*,i+1} + \beta^* \boldsymbol{w}_{j^*,i+1}, \boldsymbol{w}_{j^*,i+1} \leftarrow \alpha^* \boldsymbol{w}_{j^*,i+1}$ ▷ Out-weights
18: **return** the current DNN f'

3.2 Repair Synthesis

Next, we present how VeRe repairs the candidate neuron j^* in an iteration. Given a misclassified sample $\boldsymbol{x}$ 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 $\boldsymbol{x}$. An ideal interval should be able to effectively alleviate the wrong behavior of the DNN on $\boldsymbol{x}$ without affecting the original performance of the network.
(C2) How to modify the weights on the neuron j^* so that its activation value on $\boldsymbol{x}$ 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 $\boldsymbol{x} \in D_{\mathrm{m}}$, a target interval $s^*_{i,j^*}(\boldsymbol{x}) \in (\mathcal{I}_{i,j^*,k})_{k=1}^{K}$ is ideal, if it satisfies the following two conditions:

(1) The linear lower bound g_{i,j^*} is always positive on $s^*_{i,j^*}(\boldsymbol{x})$;
(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 $s^*_{i,j^*}(\boldsymbol{x})$. 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 $\mathrm{d}(r, \mathcal{I}) = \min_{v \in \mathcal{I}} |r - v|$ the distance between $r \in \mathbb{R}$ and an interval $\mathcal{I}$. Then, the ideal interval $s^*_{i,j^*}(\boldsymbol{x})$ can be assigned as any element in

$$\arg \min_{\mathcal{I}_{i,j^*,k} \in T} \mathrm{d}(h_{i,j^*}(\boldsymbol{x}), \mathcal{I}_{i,j^*,k}).$$

where $T = \{\mathcal{I}_{i,j^*,k} \mid \forall v \in \mathcal{I}_{i,j^*,k}\ g_{i,j^*}(v) > 0\}$ 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 the condition (1) does not hold for any $\mathcal{I}_{i,j^*,k}$. In this case, we simply choose the endpoint (of some interval $\mathcal{I}_{i,j^*,k}$) 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 $s^*_{i,j^*}(\boldsymbol{x}) = [v^*, v^*]$, where $v^* = \arg\max_{v \in \mathcal{I}_{i,j^*}} g_{i,j^*}(v)$. In Line 2–12 of Alg. 3, we show the procedure of calculating the target interval $s^*_{i,j^*}(\boldsymbol{x})$, 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. Thus, we intend to adjust the weights associated with the neuron j^* so that for as many samples $x \in D_{\mathrm{m}}$ 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 repair 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 $\tilde{\boldsymbol{w}}_{i,j^*}$ and $\tilde{b}_{i,j^*}$ as the in-weights and the bias of j^*, and two extra parameters α and β 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

$$f_{\mathrm{mini}}(h^{i-1}) = \alpha \cdot \sigma(\tilde{\boldsymbol{w}}^{\mathrm{T}}_{i,j^*} \cdot h^{i-1} + \tilde{b}_{i,j^*}) + \beta. \tag{5}$$

Through the additional linear transformation with parameters α and β, the output of this mini-network can exceed the range of the activation function σ , 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 α, and the biases of next layer are

shifted with the multiplication of β and the original corresponding out-weight of j^*, as shown in Line 16–17 of Alg. 3.

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

$$\mathcal{L}_c = \frac{1}{|D_c|} \sum_{\boldsymbol{x} \in D_c} (f_{\text{mini}}(h^{i-1}(\boldsymbol{x})) - h^i(\boldsymbol{x}))^2 ,$$
$$\mathcal{L}_m = \frac{1}{|D_m|} \sum_{\boldsymbol{x} \in D_m} (\mathrm{d}(f_{\text{mini}}(h^{i-1}(\boldsymbol{x})), s^*_{i,j^*}(\boldsymbol{x})))^2 . \quad (6)$$

Intuitively, $\mathcal{L}_c$ 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 $\mathcal{L}_m$ to guide the output of f_{mini} to move towards the ideal intervals. Specifically, when the output of f_{mini} for $h^{i-1}(\boldsymbol{x})$ is already within the ideal interval, the corresponding loss function is 0, so that no further changes for repair $\boldsymbol{x}$ are made to f_{mini}. Unlike the typical loss function that focuses on the output of the whole DNN, $\mathcal{L}$ 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 removing incorrect behavior. We use the the gradient descent algorithm to optimize the weight of f_{mini}. Specifically, we choose Adam [40] 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?
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 [28] and Blend [15], 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 $\boldsymbol{t}$ to the sample $\boldsymbol{x}$ according to the injection strategy of Blend, i.e., $\gamma \cdot \boldsymbol{t} + (1 - \gamma) \cdot \boldsymbol{x}$, where we set the ratio γ to be 0.2.

Datasets	Model	Train	Repair		Generalization	
			clean	poisoned	clean	poisoned
MNIST	CNN	60 000	1 000	1 000	5 000	5 000
CIFAR10	VGG13	50 000	1 000	1 000	5 000	5 000
SVHN	VGG13	73 257	1 000	1 000	13 016	13 016
GTSRB	VGG11	39 200	1 000	1 000	6 300	6 300
Imagenette	VGG16	9 469	200	100	1 962	1 962

Table 1: Number of the records in the datasets.

There are five datasets: MNIST [42], CIFAR-10 [41], SVHN [54], GTSRB [66] and ImageNette [32]. 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 $\widetilde{D}_r$ and $\widetilde{D}_g$, respectively. We randomly select 1 000 clean samples and 1 000 poisoned samples from D_r and $\widetilde{D}_r$ as the available data, respectively. We use D_g and $\widetilde{D}_g$ to evaluate model's generalization ability. Imagenette 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 [35, 36] networks. Each network in ACAS Xu consists of six hidden layers, each with 50 ReLU neurons. As reported in [19], 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[3] (including the property to be repaired) for each model, and sample 5 000 non-violating instances from the state space of each properties. 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[31]. 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:

$$\text{Acc} = \frac{\sum_{x \in D_g} [C_{f'}(\boldsymbol{x}) = \ell_{\boldsymbol{x}}]}{|D_g|} \quad \text{and} \quad \text{ASR} = \frac{\sum_{x \in \widetilde{D}_g} [C_{f'}(\boldsymbol{x}) = t]}{|\widetilde{D}_g|},$$

where $\ell_{\boldsymbol{x}}$ is the true label of $\boldsymbol{x}$, 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 DSR = 1 − ASR. Note that the Acc and the DSR are evaluated on Generalization set.

[3]We select the property $(2, 7, 8)$ for $N_{1,9}$, $(2, 3, 8)$ for $N_{2,9}$ and $(2, 3, 7)$ for the others.

Model	RSR on repair set/%			Generalization/%			Drawdown/%			Time/s		
	CARE	PRDNN	Ours	CARE	PRDNN	Ours	CARE	PRDNN	Ours	CARE	PRDNN	Ours
N1,9(φ_7)	**100.00**	**100.00**	**100.00**	99.98	**100.00**	**100.00**	0.01	**0.00**	**0.00**	123.31	**3.36**	5.30
N2,1(φ_2)	**100.00**	**100.00**	**100.00**	**100.00**	97.40	**100.00**	0.03	0.63	**0.02**	143.97	**3.51**	4.59
N2,2	85.56	**100.00**	**100.00**	84.88	98.35	**100.00**	33.33	29.90	**0.06**	179.79	**2.93**	3.67
N2,3	**100.00**	**100.00**	**100.00**	**100.00**	99.44	**100.00**	**0.14**	0.71	1.17	170.01	**1.98**	5.93
N2,4	62.68	**100.00**	**100.00**	61.38	99.78	**100.00**	**0.37**	33.35	33.33	135.64	2.25	**1.06**
N2,5	77.57	**100.00**	**100.00**	76.81	99.21	**99.94**	**33.33**	**33.33**	**33.33**	166.40	**2.89**	25.06
N2,6	88.04	**100.00**	**100.00**	87.88	99.48	**100.00**	**33.33**	**33.33**	**33.33**	156.60	**3.33**	3.65
N2,7	80.93	**100.00**	**100.00**	80.16	99.26	**100.00**	33.33	7.41	**0.00**	172.03	4.05	**1.35**
N2,8	**100.00**	**100.00**	**100.00**	**100.00**	97.50	**100.00**	33.33	12.15	**0.00**	122.89	3.27	**0.94**
N2,9	**100.00**	**100.00**	**100.00**	**100.00**	0.05	**100.00**	**0.00**	33.33	33.13	155.72	3.88	**1.36**
N2,9(φ_8)	99.29	**100.00**	99.95	99.31	98.37	**99.94**	**0.00**	**0.00**	0.22	168.18	23.61	**8.56**
N3,1	**100.00**	**100.00**	**100.00**	**100.00**	97.71	**100.00**	**0.00**	0.20	0.09	175.54	**5.15**	9.07
N3,2	**100.00**	**100.00**	**100.00**	**100.00**	99.54	**100.00**	33.34	5.12	**0.48**	102.52	2.10	**1.19**
N3,4	81.77	**100.00**	**100.00**	81.72	99.46	**100.00**	**33.33**	33.35	**33.33**	136.38	2.15	**1.11**
N3,5	95.74	**100.00**	**100.00**	96.12	97.52	**100.00**	**0.43**	31.07	18.50	226.59	2.56	**1.94**
N3,6	**100.00**	**100.00**	**100.00**	**100.00**	98.67	**100.00**	**0.00**	33.59	33.33	135.61	3.96	**2.09**
N3,7	96.44	**100.00**	**100.00**	96.56	91.74	**100.00**	**0.00**	33.39	33.33	238.51	3.27	**1.28**
N3,8	96.32	**100.00**	**100.00**	96.23	99.05	**100.00**	**33.33**	33.35	**33.33**	153.69	3.13	**1.13**
N3,9	99.96	**100.00**	99.64	**99.91**	95.88	99.63	33.33	33.41	**0.15**	281.20	**3.42**	36.73
N4,1	**100.00**	**100.00**	**100.00**	99.99	98.42	**100.00**	32.41	**0.00**	31.11	96.74	2.75	**1.67**
N4,3	99.75	**100.00**	99.98	99.70	99.35	**99.95**	7.76	**0.11**	0.36	94.69	**2.20**	18.15
N4,4	88.28	**100.00**	**100.00**	88.04	98.72	**100.00**	33.35	**0.05**	**0.05**	158.29	2.18	**0.96**
N4,5	99.88	**100.00**	**100.00**	99.86	99.72	**100.00**	**0.00**	0.29	**0.00**	123.66	4.10	**1.12**
N4,6	89.47	**100.00**	**100.00**	86.65	97.66	**100.00**	**33.34**	33.57	33.38	233.75	4.45	**4.37**
N4,7	99.99	**100.00**	**100.00**	99.99	98.79	**100.00**	**33.33**	**33.33**	**33.33**	124.65	3.89	**1.10**
N4,8	98.29	**100.00**	**100.00**	98.13	99.47	**99.96**	33.33	33.51	**22.24**	150.24	**3.13**	3.56
N4,9	**100.00**	**100.00**	**100.00**	**100.00**	94.37	**100.00**	33.33	30.97	**0.00**	119.46	3.69	**1.05**
N5,1	**100.00**	**100.00**	**100.00**	**100.00**	92.38	**100.00**	1.32	0.05	**0.01**	161.68	2.36	**1.97**
N5,2	**100.00**	**100.00**	**100.00**	**100.00**	92.06	**100.00**	33.33	32.99	**28.15**	111.40	3.28	**1.18**
N5,3	**100.00**	**100.00**	96.35	**100.00**	97.03	96.22	0.13	0.16	**0.11**	169.06	**2.87**	20.42
N5,4	**100.00**	**100.00**	**100.00**	**100.00**	95.98	**100.00**	**26.67**	33.36	35.10	94.87	2.13	**1.86**
N5,5	**100.00**	**100.00**	99.97	**99.96**	97.12	**99.96**	33.34	21.75	**0.27**	108.32	**2.68**	7.67
N5,6	**100.00**	**100.00**	99.96	**99.98**	99.12	99.95	**33.34**	33.39	33.37	128.95	**3.89**	6.42
N5,7	**100.00**	**100.00**	**100.00**	**100.00**	99.29	**100.00**	0.42	0.04	**0.03**	101.49	**4.23**	7.82
N5,8	**100.00**	**100.00**	**100.00**	**100.00**	99.46	**100.00**	33.34	**26.66**	33.33	114.56	3.26	**1.97**
N5,9	76.22	**100.00**	**100.00**	75.89	98.15	**100.00**	33.33	33.26	**0.00**	177.99	3.91	**3.69**
Avg	94.89	**100.00**	99.88	94.70	95.15	**99.87**	19.53	19.48	**15.26**	150.40	**3.77**	5.58

Table 2: Results of repairing violation of safety properties, where we mark the overall best values *bold*.

Attack	Dataset	Before		CARE			AI-Lancet			RM			Ours		
		Acc	ASR	Acc	DSR	Time	Acc	DSR	Time	Acc	DSR	Time	Acc	DSR	Time
BadNets	MNIST	99.70	99.02	**99.68**	99.89	393.63	**99.68**	**99.98**	140.16	95.93	99.84	**2.19**	99.27	99.83	108.40
	SVHN	93.30	99.95	84.94	99.66	709.32	81.90	97.31	2344.00	81.65	98.94	**5.27**	**92.12**	**99.90**	88.97
	CIFAR-10	82.46	99.97	77.48	99.50	920.17	80.15	99.48	1713.90	71.69	99.66	**17.89**	**81.63**	**99.85**	101.88
	GTSRB	90.87	98.91	58.60	99.82	558.58	75.49	96.92	2182.00	83.14	99.74	**7.08**	**90.32**	**99.96**	57.56
	AVG	91.58	99.46	80.18	99.72	645.43	84.30	98.42	1595.00	83.10	99.55	**8.11**	**90.84**	**99.89**	89.20
Blend	MNIST	99.72	99.97	92.80	99.93	990.24	86.82	97.28	652.80	95.01	99.58	**3.64**	**99.11**	**99.97**	62.76
	SVHN	91.23	96.45	83.08	83.31	2041.70	80.97	94.69	3859.80	84.03	**99.62**	**6.09**	**91.09**	99.43	233.61
	CIFAR-10	84.08	94.72	69.03	**99.74**	728.17	69.86	98.52	1958.00	62.05	86.42	**18.89**	**81.30**	98.96	309.20
	GTSRB	91.35	98.79	62.48	99.56	2017.71	84.94	98.86	2771.47	64.99	**99.69**	**9.30**	**90.64**	99.09	463.95
	AVG	91.60	97.48	76.85	95.63	1444.46	80.65	97.34	2310.51	76.52	96.33	**9.48**	**90.54**	**99.36**	267.38

Table 3: Results of backdoor removal, where we mark the overall best values *bold*.

For the task of fixing safety property violation, we compare VeRe with CARE, PRDNN and REASSURE [23]. For a repaired model f', the repair success rate (RSR), generalization and the drawdown can be computed as follows:

$$\text{RSR} = \frac{\sum_{x \in D_r} [C_{f'}(\boldsymbol{x}) \in P]}{|D_r|}, \quad \text{Generalization} = \frac{\sum_{x \in D_g} [C_{f'}(\boldsymbol{x}) \in P]}{|D_g|},$$

$$\text{Drawdown} = \frac{1}{3} \sum_{\varphi \in \Psi} \frac{\sum_{x \in D_\varphi} [C_{f'}(x) \in P_\varphi]}{|D_\varphi|},$$

where P is the output set of the safety property to be repaired. The set Ψ 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 ($\geq$ 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 drawdown than VeRe in one scenario, while VeRe performs better in the others. Kindly refer to [48] for more detailed results.

Backdoor Removal. For the backdoor removal task, the results of VeRe and the three baselines are presented in Table 3. All the methods achieve decent repair results on the MNIST, due to its simplicity. For more complex dataset, RM and CARE suffer from catastrophic forgetting, leading to a significant decrease in accuracy. AI-Lancet has less damage to model performance, but still results in a 2.31% to 15.38% accuracy decrease. Comparatively, VeRe has an Acc drop of 0.14% to 2.78%, which indicates that the original performance of the models has not been significantly affected.

In terms of defense success rate, our method achieves over 98% for all tasks, with the lowest being 98.96%. In comparison, the best-performing baseline, AI-Lancet, has an average DSR of 97.88%. When specifically considering the Blend attacks on CIFAR-10 and SVHN, both CARE and RM slightly outperform VeRe's in terms of defense success rate (by less than 1%). However, it is important to note that both CARE and RM methods significantly decrease the model accuracy, with CARE causing a decrease of 25.05% and RM causing a decrease of 7.2%. In contrast, our method only results in a minimal decrease in model accuracy of less than 1%. These results highlight the superiority of our method, as it not only achieves a higher defense success rate compared to the best-performing baseline, but it also minimizes the negative impact on model accuracy.

RM achieves the lowest time cost among all scenarios because it selectively fine-tunes only the problematic weights based on the gradient. In comparison, VeRe takes an average of 89.20 seconds and 267.38 seconds to remove backdoors under BadNets and Blend, respectively. On the other hand, CARE, which is based on particle swarm optimization, takes more than five times longer than our method on average. Additionally, AI-Lancet requires conducting an ablation experiment to determine the problematic weights, resulting in a larger time cost.

Answer to RQ1: VeRe is a more effective method for safety violation repairing and backdoor removal, with higher generalization/defense success rate, lower drawdown/accuracy drop, and comparable efficiency.

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

In this section, we study how does VeRe perform 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 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 VeRe remains above 98.9% in scenarios with 500+200 and 500+100 samples. In fact, even in scenarios where data is extremely limited, our method achieves an average generalization of 97.44%. In comparison, when the data size is 10 000+10 000, CARE and PRDNN only achieve a generalization of 94.70% and 95.15%, respectively. We find that as the number of available negative samples increases, the drawdown after repair with VeRe slightly increases. This is reasonable because repairing more counterexamples samples usually leads to making more significant modifications to the DNN.

Model	500+500		500+200		500+100		500+50	
	D %	G %	D %	G %	D %	G %	D %	G %
N1,9(φ_7)	0.00	100.00	0.00	99.89	0.00	99.46	0.00	98.99
N2,1(φ_2)	0.07	100.00	0.06	99.95	0.06	99.82	0.08	97.71
N2,2	11.80	100.00	2.59	100.00	0.93	99.36	0.35	98.02
N2,3	1.32	100.00	1.11	100.00	0.96	100.00	0.00	100.00
N2,4	33.33	100.00	33.33	100.00	33.33	100.00	33.33	100.00
N2,5	33.33	100.00	33.33	100.00	33.33	100.00	33.33	100.00
N2,6	33.33	100.00	33.33	100.00	33.33	100.00	33.33	100.00
N2,7	22.22	99.96	25.74	99.94	11.63	99.88	14.39	99.77
N2,8	0.00	100.00	0.00	100.00	0.00	100.00	0.00	100.00
N2,9	33.13	100.00	33.06	99.93	33.05	99.89	33.02	99.74
N2,9(φ_8)	0.00	99.98	0.00	99.98	0.00	99.98	0.00	99.97
N3,1	0.05	100.00	0.03	99.98	0.02	99.85	0.07	99.78
N3,2	0.51	99.37	0.55	98.56	0.58	96.44	0.56	93.53
N3,4	33.33	100.00	33.33	100.00	33.33	100.00	33.34	100.00
N3,5	33.33	99.95	33.33	99.99	33.33	99.74	33.33	98.94
N3,6	33.33	100.00	33.33	99.98	33.33	99.96	33.33	99.99
N3,7	33.33	99.90	33.33	99.79	33.33	99.79	33.33	99.66
N3,8	33.33	100.00	33.33	100.00	33.33	99.99	33.33	99.99
N3,9	0.05	95.77	0.06	91.51	0.03	87.27	0.02	79.01
N4,1	32.56	100.00	32.03	100.00	31.48	100.00	31.22	100.00
N4,3	0.38	99.95	0.27	99.57	0.24	98.97	0.29	97.45
N4,4	0.03	100.00	0.04	99.93	0.03	99.68	0.03	98.95
N4,5	0.00	99.97	0.00	99.91	0.00	99.82	0.00	99.79
N4,6	33.36	100.00	33.36	100.00	18.09	100.00	0.00	99.98
N4,7	33.33	100.00	33.33	100.00	33.33	100.00	33.33	99.71
N4,8	33.35	99.88	33.35	99.85	33.35	99.53	33.35	98.75
N4,9	33.33	100.00	33.33	100.00	33.33	99.96	33.33	99.79
N5,1	33.33	100.00	0.05	100.00	1.15	100.00	2.59	100.00
N5,2	33.26	100.00	33.06	100.00	32.75	100.00	32.88	99.94
N5,3	0.04	100.00	0.06	99.84	0.09	99.60	0.10	89.19
N5,4	10.18	99.97	6.71	99.71	4.84	99.64	2.89	96.98
N5,5	0.41	96.31	0.32	89.31	0.41	82.70	0.34	65.59
N5,6	33.36	99.92	33.36	99.91	33.36	99.91	33.37	99.89
N5,7	0.00	99.97	0.06	99.53	0.06	99.42	0.06	98.08
N5,8	33.33	100.00	33.33	100.00	33.33	99.64	33.33	98.51
N5,9	0.00	100.00	0.00	99.99	0.00	99.99	0.00	99.98
Avg	17.95	99.75	16.74	99.36	15.83	98.90	15.34	97.44

Table 4: Safety property repair. D: Drawdown, G: Generalization.

Subsequently, we impose constraints on the number of samples available in the backdoor removal scenario. Specifically, we fix the number of available clean samples at 1 000 and vary the number of poisoned samples among {1 000, 500, 200, 100, 50}. The results are shown in Table 5. VeRe successfully repairs poisoned models on all datasets with BadNets attacks while maintaining models' original performance . The CIFAR-10 dataset has the largest decrease in performance, ranging from 0.54% to 2.25%. Additionally, VeRe remains effective even with a small amount of available poisoned data, with DSR of the repaired models exceeding 98% in all scenarios. In comparison, AI-Lancet performs the most stably among all baselines, but causes over 10% model performance decline in the SVHN and GTSRB datasets. CARE heavily depends on the quantity of poisoned samples, and struggles to remove the backdoors stably when the number of poisoned samples is limited (with DSR of 39% on CIFAR-10 and 40% on SVHN). RM, which is based on fine-tuning, cannot maintain the original performance of the model and results in over 5% performance decline in almost all scenarios.

For Blend attacks, the performance of all baselines is not stable: AI-Lancet can not maintain original performance in any scenario; CARE suffers catastrophic forgetting on the CIFAR-10 and GTSRB datasets, leading to a significant performance decrease. Moreover, CARE is ineffective in removing backdoors on the SVHN dataset, even with 1000 available poisoned samples, resulting in a DSR of less than 90%. RM also exhibits this phenomenon on the CIFAR-10

Method	Sample	BadNets								Blend							
		MNIST		CIFAR-10		SVHN		GTSRB		MNIST		CIFAR-10		SVHN		GTSRB	
		Acc	DSR	Acc	DSR	Acc	DSR	Acc	DSR	Acc	DSR	Acc	DSR	Acc	DSR	Acc	DSR
Ours	1000+50	99.16	99.07	80.21	99.63	92.18	98.22	89.17	99.71	99.61	99.56	82.12	96.09	91.11	97.82	89.57	92.14
	1000+100	99.22	99.30	81.92	99.77	92.98	99.07	90.07	99.86	99.63	99.87	81.76	96.51	91.16	97.94	89.59	95.09
	1000+200	99.18	99.72	81.74	99.84	92.81	99.47	90.29	99.90	99.42	99.90	81.93	97.88	91.16	98.49	90.45	97.80
	1000+500	99.29	99.79	81.54	99.84	92.80	99.76	90.30	99.95	99.31	99.95	81.78	98.89	91.14	99.28	90.42	98.92
	1000+1000	99.27	99.83	81.63	99.85	92.12	99.90	90.32	99.96	99.11	99.97	81.30	98.96	91.09	99.43	90.64	99.09
AI-Lancet	1000+50	99.70	97.90	81.01	96.49	82.45	95.73	76.43	93.30	86.94	95.36	70.86	96.58	81.22	91.33	85.35	92.18
	1000+100	99.65	99.26	80.81	98.30	82.25	95.97	73.54	95.49	87.04	95.47	71.84	96.68	81.05	92.01	84.70	95.10
	1000+200	99.65	99.74	80.48	98.89	82.10	96.50	75.64	96.41	87.25	95.76	71.24	97.30	81.16	92.50	84.37	97.09
	1000+500	99.66	99.95	80.58	99.25	82.05	96.82	75.76	96.67	87.04	96.07	70.80	97.99	81.08	94.39	84.44	98.05
	1000+1000	99.68	99.89	80.15	99.48	81.90	97.31	75.49	96.92	86.82	97.28	69.86	98.52	80.97	94.69	84.94	98.86
CARE	1000+50	99.70	97.83	80.60	39.16	91.49	40.00	58.86	99.58	95.45	78.78	74.27	82.39	91.09	6.66	67.38	81.60
	1000+100	99.70	98.90	78.83	79.01	90.72	59.99	56.47	99.72	96.39	79.40	69.02	99.52	90.77	12.26	61.94	97.86
	1000+200	99.70	99.43	78.28	98.99	89.35	79.95	54.45	99.80	93.98	98.18	69.21	99.60	87.03	38.99	62.09	98.60
	1000+500	99.68	99.66	77.83	99.24	88.96	99.54	55.16	99.80	93.76	99.36	70.15	99.64	84.60	74.79	63.16	99.54
	1000+1000	99.68	99.89	77.48	99.50	84.94	99.66	58.60	99.82	93.54	99.79	69.03	99.74	83.08	83.31	62.48	99.56
RM	1000+50	97.16	99.09	79.98	88.16	61.34	97.80	78.53	92.74	83.65	92.70	65.48	31.63	66.37	92.43	60.30	92.74
	1000+100	96.01	99.67	78.10	91.46	68.47	98.09	79.83	95.83	84.19	93.50	65.32	40.78	66.81	93.32	66.41	96.62
	1000+200	93.59	99.93	75.67	94.77	78.79	98.08	78.82	96.44	81.78	95.83	64.28	56.87	77.58	96.95	56.57	98.46
	1000+500	92.45	99.94	73.45	99.50	82.35	98.84	80.24	99.17	92.48	99.04	62.80	76.70	82.64	99.17	57.12	98.99
	1000+1000	95.93	99.84	71.69	99.66	81.65	98.94	83.14	99.74	95.01	99.58	62.05	86.42	84.03	99.62	64.99	99.69

Table 5: Results of backdoor removal with limited samples, where an Acc value in color blue means that the the drop in accuracy after repair is less than 5%, and a DSR greater than 99% is highlighted in color green.

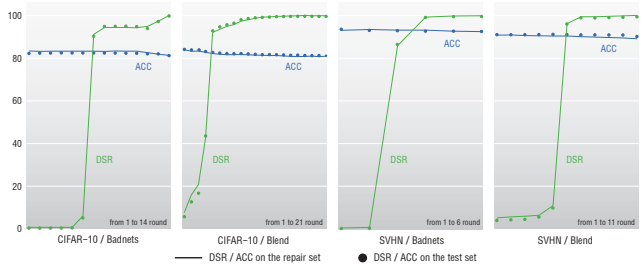

Figure 3: Repair effect after each round.

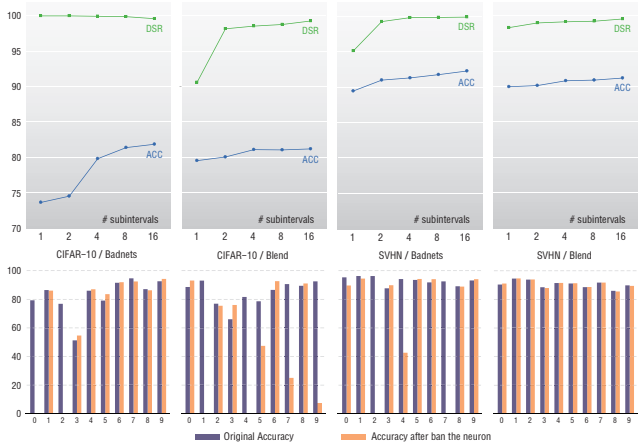

Figure 4: Effect of interval splitting and its association to frozen neuron.

dataset. Comparatively, Our method protects the original performance of the model 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 performance) 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 $\mathcal{I}_{ij}$ into K disjoint sub-intervals to reduce the approximation error. In this experiment, we study the effect of interval spliting. 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 spliting 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 the repaired neurons and freeze them in the original network. Then we compare the classification performance of the frozen model with the original model. As shown in Figure 4, the frozen repaired neurons do not significantly degrade the model's performance on the SVHN dataset under blend attacks. This indicates that these neurons may not be responsible for correctly classifying certain

Attack	Data Num	Datasets → / Methods ↓	CIFAR-10					SVHN				
			Repair Set		Generalization Set			Repair Set		Generalization Set		
			Acc	DSR	Acc	ΔAcc	DSR	Acc	DSR	Acc	ΔAcc	DSR
Badnets	1000+1000	CARE	76.23	99.69	77.48	-4.98	99.50	86.51	99.89	84.94	-8.37	99.66
		RM	77.56	99.86	71.69	-10.77	99.66	81.60	99.03	81.65	-11.66	98.94
		PSO*	68.10	100.00	67.68	-14.78	100.00	93.30	99.79	82.67	-10.64	99.42
		RM*	82.96	99.69	80.06	-2.40	98.22	90.52	100.00	89.63	-3.68	99.27
		Reg*	98.12	99.91	81.76	-0.70	97.16	98.30	99.98	92.03	-1.28	98.67
		Ours	79.50	100.00	81.63	-0.83	99.85	92.70	100.00	92.55	-0.76	99.88
	1000+50	CARE	80.12	40.00	80.60	-1.86	39.16	92.70	39.98	91.49	-1.82	40.00
		RM	79.50	90.41	79.98	-2.48	88.16	71.77	98.21	61.34	-31.97	97.80
		PSO*	70.36	80.00	70.48	-11.98	80.01	92.72	0.00	93.31	-0.00	0.05
		RM*	79.08	100.00	78.69	-3.77	93.91	90.62	100.00	90.75	-2.56	95.30
		Reg*	99.70	100.00	81.02	-1.44	88.97	99.98	100.00	92.07	-1.24	97.10
		Ours	78.60	100.00	80.21	-2.25	99.63	92.70	99.89	92.18	-1.13	98.22
Blend	1000+1000	CARE	67.90	99.90	69.03	-15.05	99.74	84.41	84.11	83.08	-8.15	83.31
		RM	68.04	94.77	62.05	-22.03	86.42	91.70	99.94	84.03	-7.20	99.62
		PSO*	58.00	100.00	58.60	-25.48	99.98	91.00	5.42	91.21	-0.02	3.95
		RM*	65.76	99.96	64.55	-19.53	96.17	83.42	100.00	80.40	-10.83	99.14
		Reg*	81.36	99.78	67.81	-16.27	96.22	98.96	100.00	85.57	-5.66	98.76
		Ours	81.48	99.60	81.30	-2.78	98.96	89.40	100.00	91.09	-0.14	99.43
	1000+50	CARE	77.44	14.16	74.27	-9.81	17.61	91.01	3.98	91.09	-0.14	6.66
		RM	66.04	39.76	65.48	-18.60	31.63	69.90	99.14	66.37	-24.86	92.43
		PSO*	58.30	100.00	59.30	-24.78	99.96	91.00	4.546	91.22	-0.01	3.62
		RM*	60.38	100.00	60.70	-23.38	92.00	74.50	100.00	74.68	-16.55	95.99
		Reg*	99.84	95.57	75.78	-8.30	89.20	99.02	100.00	83.43	-7.80	97.92
		Ours	81.28	100.00	82.12	-1.96	96.09	90.60	98.11	91.11	-0.12	97.82

Table 6: Results of combining with other repair methods on CIFAR-10 and SVHN.

categories, and thus, splitting such intervals do not lead to a significant increase in accuracy. In the other three scenarios, freezing the repaired neurons results in serious misclassification of certain categories, indicating that these neurons are responsible for both target class classification and normal category classification. Therefore, more fine-split interval can help find more ideal intervals for these neurons to protect model performance while removing the backdoor. This phenomenon matches our intuition: *interval splitting is particularly useful if the candidate neuron exposes a complex mixture of erroneous and correct behavior.* Additionally, we find that interval splitting can improve DSR in certain scenarios, resulting in performance gains of 8.68% in Blend-CIFAR-10 scenario and 4.78% in BadNets-SVHN scenario. This is because the linear relaxation without interval refinement may result in significant approximation errors, leading to mis-localization of the ideal repair interval.

Answer to RQ3: Interval splitting can reduce approximation errors, find more suitable ideal intervals, and thus more effectively repair models while preserving their original performance, which is especially the case for neurons with complex behaviors, i.e., a mixture of erroneous and correct behaviors.

4.5 The coupling between the two steps

VeRe consists of 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[29] as three baselines. We denote the replaced baselines as PSO*, RM*, and Reg* respectively. We show the results in Table 6.

Under the BadNets attack on CIFAR-10 dataset, 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 removing 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* suffered from catastrophic forgetting, resulting in over 10% performance decline. Reg* also exhibited overfitting phenomenon. By comparison, Our method protected the original performance of the model in all scenarios, with the largest decrease being 2.78%.

For the SVHN dataset, VeRe also performed best in improving DSR and maintaining Acc: with the cost of a 0.45% Acc drop, it raised the average DSR to 99.66% when the number of available negative samples was 1000; even when negative samples were scarce, VeRe could still improve DSR to 98.02%. In contrast, Reg*, which performed best in the baselines, could only improve DSR to 98.71% with 1000 available negative samples and caused a 3.47% Acc drop. When data was scarce, Reg* performed worse with an average DSR and Acc drop of 97.51% and 4.52%, respectively. Among other baselines, RM* could produce similar DSR improvements as Reg*, but caused more severe Acc drops. PSO* was ineffective in repairing in some scenarios.

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 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.

Method	BadNets			Blend		
	Acc	DSR	Time	Acc	DSR	Time
Before	83.23	0.68	-	83.54	1.87	-
CARE	82.87	2.30	3530.29	66.39	61.53	7807.44
CARE*	83.28	2.32	4476.89	57.18	100.00	13747.97
AI-Lancet	-	-	-	-	-	-
RM	74.30	84.27	7.15	63.32	63.75	9.81
Ours	81.38	98.68	122.61	78.93	96.24	207.32

Table 7: Backdoor removal results on Imagenette dataset.

4.6 Scalability

To investigate the scalability of VeRe, we conduct a backdoor removal experiment on the Imagenette dataset of high dimensional images of 224×224 pixels. 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 AI-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 overhead of CARE is unacceptable, while VeRe effectively improves the model's defense success rate at an acceptable time overhead.

To study whether VeRe generalizes to networks with other activation functions, we replace the ReLU units in each neural network with the Sigmoid activation function. Following setups in Section 4.1, we conduct a set of experiments on the CIFAR-10 and the SVHN datasets with the BadNets and the Blend attacks. Overall, VeRe can achieve an average DSR of 96.59%, while Acc decreases by up to 1.11%. The average time overhead of VeRe is 688.11s. Kindly refer to [48] for detailed results.

Answer to RQ5: VeRe is scalable to high-dimensional input and promising for other activation functions. Compared to other baselines, it improves DSR more effectively, while preserving the model's original performance.

5 RELATED WORK

DNN repair. There have been many attempts on neural network repair. Some works use heuristic algorithms such as particle swarm optimization, differential evolution, etc. DeepRepair [86] and few-shot guided mix [58] augment available negative samples to obtain new training data. Tian *et al.* [69] enhance available data by adding real-world environmental effects such as fogs to samples. To construct a provable repair, several repair methods like NNRepair, PRDNN, REASSURE [23] and ART [46] have employed formal verification techniques including constraint solving and abstract interpretation. However, the way and the purpose of employing formal verification in VeRe are quite different, which provides guidance for fault localization and target intervals for repair synthesis, and no provable guarantee is demanded. As have been stated, 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 [17] 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 [9, 21, 26, 34, 38, 39, 45, 53], abstract interpretation [25, 43, 61–63, 84], linear relaxation [5, 37, 56, 78, 83], global optimisation [20, 59, 60], CEGAR [1, 22, 55], reduction to two-player games [79, 81], and star-set abstraction [71, 72]. These methods provide provable estimates of DNN robustness. Besides, statistical methods like [3, 4, 11, 12, 33, 44, 50, 76, 77, 80] are more efficient and scalable for complex DNN structures, where quantitative robustness is provable at a certain confidence level. In VeRe, we employ CROWN [83] as the verification engine for repair synthesis, whilst other tools like ERAN [65], Fast-lin [78], DeepSymbol [43] are also adoptable.

6 CONCLUSION

We propose VeRe, a novel verification guided synthesis framework for repairing DNNs. VeRe performs linear relaxation on fully connected layer to localize problematic neurons and provide the target interval for repair, which guides us to construct an optimization problem for the optimal repair. We conduct an empirical evaluation using five image classification datasets and one safety property dataset. 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.

ACKNOWLEDGMENTS

This work is supported by the Key R&D Programs of Zhejiang (2022C01018), the National Natural Science Foundation of China (62102359, 62176080), the CAS Project for Young Scientists in Basic Research (Grant No.YSBR-040), and the Joint Funds of National Natural Science Foundation of China (U21B2001).

REFERENCES

[1] Pranav Ashok, Vahid Hashemi, Jan Kretínský, and Stefanie Mohr. 2020. DeepAbstract: Neural Network Abstraction for Accelerating Verification. In *ATVA 2020 (Lecture Notes in Computer Science, Vol. 12302)*. Springer, 92–107.

[2] Maryam Badar, Muhammad Haris, and Anam Fatima. 2020. Application of deep learning for retinal image analysis: A review. *Computer Science Review* 35 (2020), 100203.

[3] Teodora Baluta, Zheng Leong Chua, Kuldeep S Meel, and Prateek Saxena. 2021. Scalable quantitative verification for deep neural networks. In *ICSE 2021*. IEEE, Madrid, Spain, 312–323.

[4] Teodora Baluta, Shiqi Shen, Shweta Shinde, Kuldeep S. Meel, and Prateek Saxena. 2019. Quantitative Verification of Neural Networks and Its Security Applications. In *CCS 2019, November 11-15, 2019*. ACM, London, UK, 1249–1264.

[5] Ben Batten, Panagiotis Kouvaros, Alessio Lomuscio, and Yang Zheng. 2021. Efficient neural network verification via layer-based semidefinite relaxations and linear cuts. IJCAI.

[6] Per Bjesse. 2005. What is formal verification? *ACM SIGDA Newsletter* 35, 24 (2005), 1–es.

[7] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[9] Rudy Bunel, P Mudigonda, Ilker Turkaslan, P Torr, Jingyue Lu, and Pushmeet Kohli. 2020. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research* 21, 2020 (2020).

[10] Gabriel Cadamuro, Ran Gilad-Bachrach, and Xiaojin Zhu. 2016. Debugging machine learning models. In *ICML Workshop on Reliable Machine Learning in the Wild*, Vol. 103.

[11] Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, Nicola Paoletti, Andrea Patane, and Matthew Wicker. 2019. Statistical Guarantees for the Robustness of Bayesian Neural Networks. In *IJCAI 2019, August 10-16, 2019*, Sarit Kraus (Ed.). ijcai.org, Macao, China, 5693–5700.

[12] Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, and Andrea Patane. 2019. Robustness Guarantees for Bayesian Inference with Gaussian Processes. In *AAAI 2019, January 27 - February 1, 2019*. AAAI Press, Honolulu, Hawaii, USA, 7759–7768.

[13] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*. Ieee, 39–57.

[14] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. 2018. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv:1811.03728* (2018).

[15] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526* (2017).

[16] Wenyuan Dai, Ou Jin, Gui-Rong Xue, Qiang Yang, and Yong Yu. 2009. Eigentransfer: a unified framework for transfer learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*. 193–200.

[17] Cristina David and Daniel Kroening. 2017. Program synthesis: challenges and opportunities. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 375, 2104 (2017), 20150403.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[19] Guoliang Dong, Jun Sun, Xingen Wang, Xinyu Wang, and Ting Dai. 2021. Towards repairing neural networks correctly. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 714–725.

[20] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. 2018. Output Range Analysis for Deep Feedforward Neural Networks. In *NFM 2018 (Lecture Notes in Computer Science, Vol. 10811)*, Aaron Dutle, César A. Muñoz, and Anthony Narkawicz (Eds.). Springer, Newport News, VA, USA, 121–138.

[21] Rüdiger Ehlers. 2017. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *ATVA 2017*. Springer, Pune, India, 269–286.

[22] Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. 2020. An Abstraction-Based Framework for Neural Network Verification. In *CAV 2020 (Lecture Notes in Computer Science, Vol. 12224)*, Shuvendu K. Lahiri and Chao Wang (Eds.). Springer, Los Angeles, CA, USA, 43–65.

[23] Feisi Fu and Wenchao Li. 2022. Sound and Complete Neural Network Repair with Minimality and Locality Guarantees. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net. https://openreview.net/forum?id=xS8AMYiEav3

[24] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-adversarial training of neural networks. *The journal of machine learning research* 17, 1 (2016), 2096–2030.

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

[26] Sumathi Gokulanathan, Alexander Feldsher, Adi Malca, Clark Barrett, and Guy Katz. 2020. Simplifying neural networks using formal verification. In *NASA Formal Methods: 12th International Symposium, NFM 2020, Moffett Field, CA, USA, May 11–15, 2020, Proceedings 12*. Springer, 85–93.

[27] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[28] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2019. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access* 7 (2019), 47230–47244.

[29] Dario Guidotti, Luca Pulina, and Armando Tacchella. 2020. Never 2.0: Learning, verification and repair of deep neural networks. *arXiv preprint arXiv:2011.09933* (2020).

[30] Dan Hendrycks and Thomas Dietterich. 2019. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261* (2019).

[31] Patrick Henriksen, Francesco Leofante, and Alessio Lomuscio. 2022. Repairing misclassifications in neural networks using limited data. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*. 1031–1038.

[32] Jeremy Howard. 2019. The Imagenette dataset. https://github.com/fastai/imagenette

[33] Pei Huang, Yuting Yang, Minghao Liu, Fuqi Jia, Feifei Ma, and Jian Zhang. 2021. ϵ-weakened Robustness of Deep Neural Networks. *CoRR* abs/2110.15764 (2021). arXiv:2110.15764

[34] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 2017. Safety Verification of Deep Neural Networks. In *CAV 2017*. Springer, Heidelberg, Germany, 3–29.

[35] Kyle D Julian, Mykel J Kochenderfer, and Michael P Owen. 2019. Deep neural network compression for aircraft collision avoidance systems. *Journal of Guidance, Control, and Dynamics* 42, 3 (2019), 598–608.

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

[37] Kyle D Julian, Shivam Sharma, Jean-Baptiste Jeannin, and Mykel J Kochenderfer. 2019. Verifying aircraft collision avoidance neural networks through linear approximations of safe regions. *arXiv preprint arXiv:1903.00762* (2019).

[38] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. 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*. Springer, 97–117.

[39] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *CAV 2019 (Lecture Notes in Computer Science, Vol. 11561)*, Isil Dillig and Serdar Tasiran (Eds.). Springer, New York City, NY, USA, 443–452.

[40] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[41] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[42] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[43] Jianlin Li, Jiangchao Liu, Pengfei Yang, Liqian Chen, Xiaowei Huang, and Lijun Zhang. 2019. 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*. Springer, 296–319.

[44] Renjue Li, Pengfei Yang, Cheng-Chao Huang, Youcheng Sun, Bai Xue, and Lijun Zhang. 2022. Towards Practical Robustness Analysis for DNNs based on PAC-Model Learning. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 2189–2201. https://doi.org/10.1145/3510003.3510143

[45] Wang Lin, Zhengfeng Yang, Xin Chen, Qingye Zhao, Xiangkun Li, Zhiming Liu, and Jifeng He. 2019. Robustness verification of classification deep neural networks via linear programming. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11418–11427.

[46] Xuankang Lin, He Zhu, Roopsha Samanta, and Suresh Jagannathan. 2020. ART: abstraction refinement-guided training for provably correct neural networks. In *# PLACEHOLDER_PARENT_METADATA_VALUE#*, Vol. 1. TU Wien Academic Press, 148–157.

[47] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2018. Trojaning Attack on Neural Networks. In *NDSS*.

[48] Jianan Ma, Pengfei Yang, Jingyi Wang, Youcheng Sun, Cheng-chao Huang, and Zhen Wang. 2023. VERE. https://github.com/nninjn/VeRe

[49] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. 2018. MODE: automated neural network model debugging via state differential analysis and input selection. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 175–186.

[50] Ravi Mangal, Aditya V. Nori, and Alessandro Orso. 2019. Robustness of neural networks: a probabilistic and practical approach. In *ICSE (NIER) 2019, Montreal, QC, Canada, May 29-31, 2019*. IEEE / ACM, Montreal, QC, Canada, 93–96.

[51] Mark Huasong Meng, Guangdong Bai, Sin Gee Teo, Zhe Hou, Yan Xiao, Yun Lin, and Jin Song Dong. 2022. Adversarial robustness of deep neural networks: A survey from a formal verification perspective. *IEEE Transactions on Dependable and Secure Computing* (2022).

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

[53] Nina Narodytska, Shiva Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. 2018. Verifying properties of binarized deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[54] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. 2011. Reading digits in natural images with unsupervised feature learning. (2011).

[55] Matan Ostrovsky, Clark W. Barrett, and Guy Katz. 2022. An Abstraction-Refinement Approach to Verifying Convolutional Neural Networks. In *Automated Technology for Verification and Analysis - 20th International Symposium, ATVA 2022, Virtual Event, October 25-28, 2022, Proceedings (Lecture Notes in Computer Science, Vol. 13505)*. Springer, 391–396.

[56] Brandon Paulsen, Jingbo Wang, and Chao Wang. 2020. Reludiff: Differential verification of deep neural networks. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 714–726.

[57] Luca Pulina and Armando Tacchella. 2012. Challenging SMT solvers to verify neural networks. *Ai Communications* 25, 2 (2012), 117–135.

[58] Xuhong Ren, Bing Yu, Hua Qi, Felix Juefei-Xu, Zhuo Li, Wanli Xue, Lei Ma, and Jianjun Zhao. 2020. Few-shot guided mix for dnn repairing. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 717–721.

[59] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. 2018. Reachability Analysis of Deep Neural Networks with Provable Guarantees. In *IJCAI 2018*. ijcai.org, Stockholm, Sweden, 2651–2659.

[60] Wenjie Ruan, Min Wu, Youcheng Sun, Xiaowei Huang, Daniel Kroening, and Marta Kwiatkowska. 2019. Global Robustness Evaluation of Deep Neural Networks with Provable Guarantees for the Hamming Distance. In *IJCAI 2019*, Sarit Kraus (Ed.). ijcai.org, Macao, China, 5944–5952.

[61] Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin T. Vechev. 2019. Beyond the Single Neuron Convex Barrier for Neural Network Certification. In *NeurIPS 2019*. 15072–15083.

[62] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. 2018. Fast and effective robustness certification. *Advances in neural information processing systems* 31 (2018).

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

[64] Jeongju Sohn, Sungmin Kang, and Shin Yoo. 2022. Arachne: Search Based Repair of Deep Neural Networks. *ACM Transactions on Software Engineering and Methodology* (2022).

[65] ETH Zurich SRI Lab, Department of Computer Science. 2020. ETH Robustness Analyzer for Neural Networks (ERAN). https://github.com/eth-sri/eran

[66] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks* 32 (2012), 323–332.

[67] Bing Sun, Jun Sun, Long H Pham, and Jie Shi. 2022. Causality-based neural network repair. In *Proceedings of the 44th International Conference on Software Engineering*. 338–349.

[68] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. 2018. Testing deep neural networks. *arXiv preprint arXiv:1803.04792* (2018).

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

[70] Florian Tramer and Dan Boneh. 2019. Adversarial training and robustness for multiple perturbations. *Advances in neural information processing systems* 32 (2019).

[71] Hoang-Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T. Johnson. 2020. Verification of Deep Convolutional Neural Networks Using ImageStars. In *CAV 2020 (Lecture Notes in Computer Science, Vol. 12224)*, Shuvendu K. Lahiri and Chao Wang (Eds.). Springer, Los Angeles, CA, USA, 18–42.

[72] Hoang-Dung Tran, Diego Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T. Johnson. 2019. Star-Based Reachability Analysis of Deep Neural Networks. In *FM 2019 (Lecture Notes in Computer Science, Vol. 11800)*, Maurice H. ter Beek, Annabelle McIver, and José N. Oliveira (Eds.). Springer, Porto, Portugal, 670–686.

[73] Muhammad Usman, Divya Gopinath, Youcheng Sun, and Corina S Păsăreanu. 2022. Rule-Based Runtime Mitigation Against Poison Attacks on Neural Networks. In *Runtime Verification*. Springer, 67–84.

[74] Sandra Vieira, Walter HL Pinaya, and Andrea Mechelli. 2017. Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications. *Neuroscience & Biobehavioral Reviews* 74 (2017), 58–75.

[75] Yisen Wang, Xuejiao Deng, Songbai Pu, and Zhiheng Huang. 2017. Residual convolutional CTC networks for automatic speech recognition. *arXiv preprint arXiv:1702.07793* (2017).

[76] Stefan Webb, Tom Rainforth, Yee Whye Teh, and M. Pawan Kumar. 2019. A Statistical Approach to Assessing Neural Network Robustness. In *ICLR 2019*. OpenReview.net, New Orleans, LA, USA.

[77] Lily Weng, Pin-Yu Chen, Lam M. Nguyen, Mark S. Squillante, Akhilan Boopathy, Ivan V. Oseledets, and Luca Daniel. 2019. PROVEN: Verifying Robustness of Neural Networks with a Probabilistic Approach. In *ICML 2019, 9-15 June 2019 (Proceedings of Machine Learning Research, Vol. 97)*. PMLR, Long Beach, California, USA, 6727–6736.

[78] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane S. Boning, and Inderjit S. Dhillon. 2018. Towards Fast Computation of Certified Robustness for ReLU Networks. In *ICML 2018 (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, Stockholm, Sweden, 5273–5282.

[79] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. 2018. Feature-Guided Black-Box Safety Testing of Deep Neural Networks. In *TACAS 2018 (Lecture Notes in Computer Science, Vol. 10805)*, Dirk Beyer and Marieke Huisman (Eds.). Springer, Thessaloniki, Greece, 408–426.

[80] Matthew Wicker, Luca Laurenti, Andrea Patane, and Marta Kwiatkowska. 2020. Probabilistic Safety for Bayesian Neural Networks. In *UAI 2020, August 3-6, 2020 (Proceedings of Machine Learning Research, Vol. 124)*, Ryan P. Adams and Vibhav Gogate (Eds.). AUAI Press, virtual online, 1198–1207.

[81] Min Wu, Matthew Wicker, Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. 2020. A game-based approximate verification of deep neural networks with provable guarantees. *Theor. Comput. Sci.* 807 (2020), 298–329.

[82] Ruihan Wu, Chuan Guo, Yi Su, and Kilian Q Weinberger. 2021. Online adaptation to label distribution shift. *Advances in Neural Information Processing Systems* 34 (2021), 11340–11351.

[83] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. 2020. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems* 33 (2020), 1129–1141.

[84] Pengfei Yang, Renjue Li, Jianlin Li, Cheng-Chao Huang, Jingyi Wang, Jun Sun, Bai Xue, and Lijun Zhang. 2021. Improving Neural Network Verification through Spurious Region Guided Refinement. In *TACAS 2021 (Lecture Notes in Computer Science, Vol. 12651)*. Springer, 389–408.

[85] Wei Ying, Yu Zhang, Junzhou Huang, and Qiang Yang. 2018. Transfer learning via learning to transfer. In *International conference on machine learning*. PMLR, 5085–5094.

[86] Bing Yu, Hua Qi, Qing Guo, Felix Juefei-Xu, Xiaofei Xie, Lei Ma, and Jianjun Zhao. 2021. Deeprepair: Style-guided repairing for deep neural networks in the real-world operational environment. *IEEE Transactions on Reliability* 71, 4 (2021), 1401–1416.

[87] Hao Zhang and WK Chan. 2019. Apricot: A weight-adaptation approach to fixing deep learning models. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 376–387.

[88] Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane Boning, and Cho-Jui Hsieh. 2019. Towards stable and efficient training of verifiably robust neural networks. *arXiv preprint arXiv:1906.06316* (2019).

[89] Yue Zhao, Hong Zhu, Kai Chen, and Shengzhi Zhang. 2021. Ai-lancet: Locating error-inducing neurons to optimize neural networks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 141–158.