



Patch Synthesis for Property Repair of Deep Neural Networks

Zhiming Chi^{1,2}, Jianan Ma^{3,4}, Pengfei Yang^{5*}, Cheng-Chao Huang⁶, Renjue Li^{1,2},
Jingyi Wang⁴, Xiaowei Huang⁷ and Lijun Zhang¹

¹Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

²University of Chinese Academy of Sciences, Beijing, China

³School of cyberspace, Hangzhou Dianzi University, Hangzhou, China

⁴Zhejiang University, Hangzhou, China

⁵College of Computer and Information Science, Software College, Southwest University, Chongqing, China

⁶Nanjing Institute of Software Technology, Chinese Academy of Sciences, Nanjing, China

⁷University of Liverpool, Liverpool, United Kingdom

Email: chizm@ios.ac.cn, majianannn@gmail.com, ypfbest001@swu.edu.cn, chengchao@njis.ac.cn,

lirj19@ios.ac.cn, wangjyee@zju.edu.cn, xiaowei.huang@liverpool.ac.uk, zhanglj@ios.ac.cn

Abstract—Deep neural networks (DNNs) are prone to various dependability issues, such as adversarial attacks, which hinder their adoption in safety-critical domains. Recently, NN repair techniques have been proposed to address these issues while preserving original performance by locating and modifying guilty neurons and their parameters. However, existing repair approaches are often limited to specific data sets and do not provide theoretical guarantees for the effectiveness of the repairs. To address these limitations, we introduce PATCHPRO, a novel patch-based approach for property-level repair of DNNs, focusing on local robustness. The key idea behind PATCHPRO is to construct patch modules that, when integrated with the original network, provide specialized repairs for all samples within the robustness neighborhood while maintaining the network's original performance. Our method incorporates formal verification and a heuristic mechanism for allocating patch modules, enabling it to defend against adversarial attacks and generalize to other inputs. PATCHPRO demonstrates superior efficiency, scalability, and repair success rates compared to existing DNN repair methods, i.e., realizing provable property-level repair for 100% cases across multiple high-dimensional datasets.

I. INTRODUCTION

In recent years, deep neural networks (DNNs) have achieved significant advancements in various domains, including computer vision [1], natural language processing [2], and speech recognition [3]. Despite these advancements, the adoption of DNNs in safety-critical domains has been slow due to concerns regarding their dependability. A major concern is the vulnerability of DNNs to adversarial attacks [4], [5], where adversaries can manipulate input data in ways that are imperceptible to humans but can cause the model to make incorrect decisions. This vulnerability poses serious safety risks in applications such as autonomous vehicles [6] and medical diagnosis [7]. Therefore, it is crucial to regularly update the DNN to mitigate the risks associated with these errors and ensure the reliability of the network in practical applications.

DNN repair techniques [8]–[10] have been proposed to address the problem involving rectifying errors in the network by modifying its architecture or parameters. Compared to traditional methods such as adversarial training [11]–[13], input sanitization, fine-tuning, transfer learning [14], [15], and data augmentation [16]–[18], neuron-level fault localization and repair methods [8] offer a more targeted approach by identifying and correcting errors at the individual neuron level while not affecting the overall network performance.

Despite significant advancements in this field, several limitations remain. First, neuron-level repair techniques, which rely on a limited number of samples, often struggle to provide robust defense against adversarial attacks due to the inherent complexity of these attacks compared to simpler threats like backdoor attacks. Adversarial attacks involve intricate mixtures of features [19], making it difficult to generalize parameter adjustments from a small dataset. Second, existing repair methods typically focus on specific data for repair and fail to generalize to the property level (e.g., local robustness), which limits their effectiveness in addressing a broad range of adversarial scenarios. This motivates us to explore property repair of DNNs, specifically to fix the safety properties that neural networks violate in certain input regions. Third, while provable repair methods such as PRDNN [20], REASSURE [21], and APRNN [22] can conduct property-based error correction, they achieve this by ensuring that outputs meet constraints on the vertices of the input region as a polyhedron. However, the number of vertices increases exponentially with data dimensionality, which makes these methods inefficient for high-dimensional data.

In this work, we aim to address these limitations by proposing a novel patch-based method to achieve provable repair on the property level. Specifically, we focus on correcting potential adversarial samples in the infinite set of high-dimensional points within a certain neighborhood of these error samples, i.e., satisfying local robustness property. To achieve this, our

*is the corresponding author.

key idea is to use formal verification to help construct a separate patch module (in the form of a fully connected neural network structure) for each neighborhood outside of the original network. Such a patch-based method allows us not only to ensure that the infinite set of points within the error sample’s neighborhood repaired but also to maintain the performance of the original network unaffected. To construct such a patch, we utilize reachability analysis through linear relaxation for the provable training of these patch modules. Specifically, we use the verification method DeepPoly [23] to create a linear relaxation of the output neurons. This approximation is employed to determine the distance between the targeted behavior and the current behavior, serving as the loss function. The patch modules are then trained to minimize this distance. Once the loss function reaches zero, the patch modules will offer provable repairs for adversarial attacks within the perturbation region.

To ensure that each patch module addresses specific neighborhoods, our approach employs an external indicator that identifies inputs within a particular sample’s local neighborhood and assigns the appropriate patch modules for repair. This indicator allows the same patch module to effectively repair adversarial attacks within the same perturbation region. For adversarial samples outside the repaired property, the indicator utilizes a heuristic allocation mechanism to assign suitable patch modules, thereby enhancing the generalization of the repair. This comprehensive framework not only improves the network’s resilience against adversarial attacks but also maintains its accuracy. By leveraging reachability analysis and dedicated patch modules, we provide provable repairs for adversarial samples, significantly boosting the network’s robustness. Additionally, to extend this local robustness across the entire dataset, we integrate the external indicator with the original network, ensuring that all samples, including those beyond the initially repaired property, benefit from heuristic patch module allocation. This approach effectively combines local and global repair strategies to enhance overall network robustness.

We summarize our contributions as follows:

- We introduce PATCHPRO, a novel approach that integrates a loss function derived from formal verification to train patch modules within the original network. This method effectively repairs high-dimensional infinite point sets within the polyhedron neighborhood of adversarial samples while preserving the network’s performance. The approach also includes a heuristic allocation mechanism, which ensures the generalization of local robustness repair by seamlessly integrating patch modules into the original network architecture.
- To tackle the efficiency challenges of formal verification in large-scale DNNs, we utilize patch modules to perform repairs in the feature space of the networks. This strategy enables our method to scale effectively across various network architectures, ensuring both efficient and practical implementation.
- We thoroughly evaluate PATCHPRO on three diverse

datasets and multiple DNN architectures. Through extensive comparisons with state-of-the-art repair and adversarial training techniques, our method consistently demonstrates superior efficiency, scalability, and generalization capabilities. It shows significant improvement in handling general inputs, thereby greatly enhancing the overall robustness of the network.

II. PRELIMINARY

In this section we recall some basic notations of DNN repair. A deep neural network is a function $N: \mathbb{R}^{n_o} \rightarrow \mathbb{R}^{n_L}$ that maps an input $x \in \mathbb{R}^{n_o}$ to an output $y \in \mathbb{R}^{n_L}$. We usually visualize a DNN N as a sequence of L layers, where the i th layer contains n_i neurons each representing a real variable. Between two adjacent layers is typically a composition of an affine function and a non-linear activation function, and the DNN N is the composition of the functions between layers. In many applications, DNNs are serving for classification tasks. In such a classification DNN $N: \mathbb{R}^{n_o} \rightarrow \mathbb{R}^{n_L}$, every output dimension corresponds to a classification label, and the one with the maximum output value is the classification result that the DNN N gives, i.e., $C_N(x) = \arg \max_{1 \leq i \leq n_L} N(x)_i$, where $N(x)_i$ is the i th entry of the vector $N(x)$.

The notion of safety properties pertains to assertions that guarantee the absence of undesirable behavior. Within the context of DNNs, a safety property demands that a DNN operates correctly within a specified input range.

Definition 1. A safety property is a triple (N, X, Q) , where N is a DNN, $X \subseteq \mathbb{R}^{d_{in}}$ and $Q \subseteq \mathbb{R}^{d_{out}}$ are the subset of input and output spaces of the neural network N . The property (N, X, Q) is satisfied if and only if $N(x) \in Q$ for all $x \in X$.

A local robustness property of a classification DNN N requires that for any input x in a given neighborhood $B(x_0, r)$ of an input x_0 , its classification should always be consistent with x_0 , where a neighborhood of an input x_0 is usually defined as a closed ball $B(x_0, r) := \{x \in \mathbb{R}^{n_o} \mid \|x - x_0\|_\infty \leq r\}$, $\|\cdot\|_\infty$ is the L_∞ -norm, and $r > 0$ is the radius. Formally, it can be defined as follows:

The local robustness of a DNN refers to its ability to maintain stable and consistent predictions in the vicinity of its in-distribution data points, even in the presence of small perturbations or variations in the input. Here a neighborhood of an input x_0 is usually defined as a closed ball $B(x_0, r) := \{x \in \mathbb{R}^{n_o} \mid \|x - x_0\|_\infty \leq r\}$, where $\|\cdot\|_\infty$ is the L_∞ -norm, and $r > 0$ is the radius. Formally, a local robustness property of a classification DNN N requires that for any input x in a given neighborhood $B(x_0, r)$ of an input x_0 , its classification should always be consistent with x_0 , i.e., $\forall x \in B(x_0, r), C_N(x) = C_N(x_0)$. We denote this local robustness property as $(N, B(x_0, r))$, i.e. $(N, B(x_0, r)) = (N, B(x_0, r), \{y \in \mathbb{R}^{n_L} \mid y_i < y_{C_N(x_0)}, i = 1, 2, \dots, n_L\})$, and thus it is a safety property.

In this work, we focus on the problem of repairing adversarial attacks with limited adversarial samples. Different from repairing backdoor attacks, not only does the buggy

behavior of the given adversarial samples need fixing, but we also require that there should be no adversarial attacks around given adversarial samples, and that this enhancement of local robustness should generalize to samples across the whole dataset. Now we formally state the problem of repairing adversarial attacks as follows:

Given a DNN N and a set of adversarial samples $\{x_i^*\}_{i=1}^n$, where n can be significantly smaller than the size of the training set, and each x_i^* is obtained by an adversarial attack on an input x_i with a given radius r , we need to construct a DNN F which is locally robust on every $B(x_i, r)$ while the accuracy is maintained and local robustness of other inputs is potentially improved.

III. METHODOLOGY

We focus on fixing adversarial attacks with limited data in this work, and the aims of the repair are at least threefold: For a given radius $r > 0$,

- the buggy behaviors of $\{x_i^*\}_{i=1}^n$ are fixed,
- the DNN is locally robust in $B(x_i, r)$ for $1 \leq i \leq n$,
- the accuracy of the DNN is maintained, and for as many input x in the dataset as possible, the DNN is locally robust in $B(x, r)$.

In this work, we propose a patch-based repair method. A patch refers to a specific modification or alteration made to a software system or codebase; it is a discrete set of changes applied to fix a bug, enhance functionality, or address security vulnerabilities. Patch-based DNN repair involves the integration of an external indicator designed to identify buggy inputs, followed by the application of specialized patch modules to repair input sets which have similar behaviors. Each indicator here corresponds precisely to a robustness neighborhood $B(x_i, r)$ that requires repair, and we can leverage verification tools to ensure that the repair within each robustness neighborhood is provable. Furthermore, for other inputs in the dataset, we can heuristically match patches that maximize their robustness, thereby endowing this repair approach with global generalization. In this section, we propose a patch-based repair method named PATCHPRO to defend from adversarial attacks with limited data.

A. Structure of the repaired DNN

The main produce of the repaired network is shown in Fig. 1. For an input $x \in \mathbb{R}^{n_0}$, the role of the indicator is to select the appropriate patches that, when applied, yields the sum of the outputs of the patch modules and the output of the original DNN for x . This sum represents the output obtained after the repair process. The indicator function is defined as:

Definition 2. Let $\mathcal{C} = (X_1, \dots, X_m)$ be a finite sequence of input properties. The indicator function $\mathcal{I}_{\mathcal{C}}: \mathbb{R}^{n_0} \rightarrow \{0, 1\}^m$ outputs in the j th entry, where $j \in \{1, \dots, m\}$, as

$$\mathcal{I}_{\mathcal{C}}(x)_j = \begin{cases} 1, & \text{if } x \models X_j, \\ 0, & \text{otherwise.} \end{cases}$$

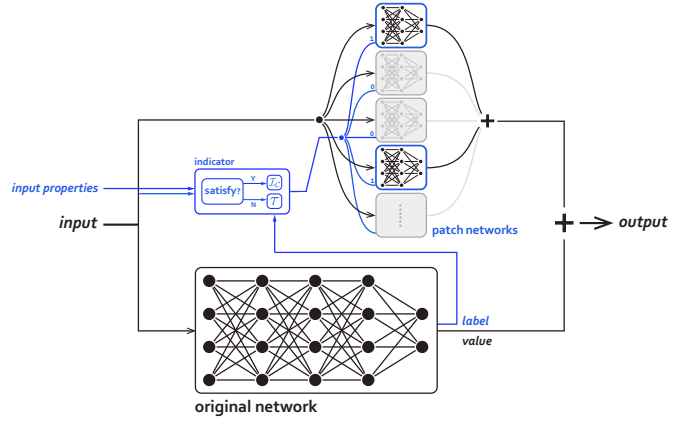


Fig. 1. The architecture of a DNN repaired by PATCHPRO. It contains multiple patch networks for each input properties. Each of them is enabled or disabled according to the allocation signal “1” or “0” determined by the indicator. The blue lines highlights the indicator’s workflow. The final output is the sum of the outputs of all the enabled patches and the original network.

Typically an input property X_i is a subset of \mathbb{R}^{n_0} , and we define $x \models X_i$ iff $x \in X_i$.

Upon classification by the indicator, a set of patch modules is deployed to perform the repair. Here each patch module is specifically tailored to address an input set with the same local robustness property. The implementation of a patch module is a fully connected neural network in this work. For an input x that does not satisfy any input properties, i.e., $x \notin \bigcup_{i=1}^n B(x_i, r)$, there is no existing specific patch module, but it may still suffer from adversarial attacks within its neighborhood. In this situation, we heuristically allocate some patch modules to this input to defend from adversarial attacks. Formally, the structure of the repaired DNN is as follows:

Definition 3. A repaired DNN is a tuple $F = (N, \mathcal{C}, \mathcal{P}, \tau)$, where

- $N: \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_L}$ is the original DNN,
- $\mathcal{C} = (X_1, \dots, X_m)$ be a finite sequence of input properties,
- $\mathcal{P} = (P_1, \dots, P_m)^T$ is a finite sequence of patch modules, each of which is a fully connected neural network,
- and $\tau: \mathbb{R}^{n_0} \setminus \bigcup_{i=1}^n X_i \rightarrow 2^{\{1, \dots, m\}}$ is a patch allocation function.

The semantics of the repaired DNN F is a function:

$$F: \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_L},$$

$$x \mapsto \begin{cases} N(x) + \mathcal{I}_{\mathcal{C}}(x)^T \mathcal{P}(x), & \text{if } x \in \bigcup_{i=1}^n X_i, \\ N(x) + \sum_{j \in \tau(x)} P_j(x), & \text{otherwise,} \end{cases}$$

where $\mathcal{P}(x) = (P_1(x), \dots, P_m(x))^T$.

In this work, we set $\mathcal{C} = \{B(x_i, r) \mid i = 1, \dots, n\}$, i.e., the indicator judges which robustness region the input belongs to, and the corresponding patch module defends against adversarial attacks in this specific region.

B. Training the patch modules

The first challenge is to train the patch modules, which are each implemented as a fully connected neural network, to repair each robustness property $(F, B(x_i, r))$ with a provable guarantee. To achieve provability, we employ formal verification in training these patch modules. The result of verification on a robustness property can be transformed into a loss function; once this loss reaches zero, the property is verified to be true, and the patch module in this stage can defend against all possible adversarial attacks in this robustness region. This idea has been widely adopted in DNN repair [24], [25].

In this work, we employ DeepPoly [23] as the verification engine¹. DeepPoly uses abstract interpretation to give every neuron an upper/lower bound in the form of an affine function, where only variables in previous layers occur, and the numerical bound can be derived by propagating backwards these affine upper/lower bounds to the input layer. Recall that a local robustness property $(F, B(x_i, r))$ holds iff for any $x \in B(x_i, r)$ and $\ell \neq \ell_0 = C_F(x_i)$, $F(x)_\ell < F(x)_{\ell_0}$, so DeepPoly calculates the abstraction of F on $B(x_i, r)$ for every neuron and the expressions $F(x)_\ell - F(x)_{\ell_0}$ for $\ell \neq \ell_0$. From the abstraction, we can obtain an affine function in which there are only input variables as a sound upper bound of $F(x)_\ell - F(x)_{\ell_0}$ on $B(x_i, r)$, i.e.,

$$\forall x \in B(x_i, r), F(x)_\ell - F(x)_{\ell_0} \leq \alpha_\ell^T x + \beta_\ell, \quad (1)$$

where $\alpha_\ell \in \mathbb{R}^{n_0}$ and $\beta_\ell \in \mathbb{R}$ are constants. It is easy to obtain the numerical upper bound of $\alpha_\ell^T x + \beta_\ell$ on the box region $B(x_i, r)$. If this upper bound is negative for every $\ell \neq \ell_0$, then the local robustness property $(F, B(x_i, r))$ is verified to be true by DeepPoly. Therefore, it is natural to use this upper bound as the loss function to train the corresponding patch module.

Definition 4. For a local robustness property $\varphi = (F, B(x_i, r))$, we define the safety violated loss function as

$$\mathcal{L}(\varphi)(x) = \sum_{\ell \neq \ell_0} \max(\alpha_\ell^T x + \beta_\ell, 0),$$

where $\alpha_\ell^T x + \beta_\ell$ is the upper bound of $F(x)_\ell - F(x)_{\ell_0}$ on $B(x_i, r)$ given by DeepPoly, as shown in Eq. (1). For local robustness properties $\varphi_1, \dots, \varphi_k$ which share the same ground truth label ℓ_0 , we define $\mathcal{L}(\bigwedge_{i=1}^k \varphi_i) = \sum_{i=1}^k \mathcal{L}(\varphi_i)$.

For a local robustness property $\varphi = (F, B(x_i, r))$, its safety violated loss function $\mathcal{L}(\varphi)$ being 0 implies that φ is verified to be true by DeepPoly.

Theorem 1. Let $\varphi = (F, B(x_i, r))$ be a local robustness property. If $\mathcal{L}(\varphi) = 0$ on $B(x_i, r)$, i.e.,

$$\mathcal{L}^*(\varphi) := \max(\text{elmax}(\alpha_\ell^T, \mathbf{0}) \cdot (x_i + r \cdot \mathbf{1}) + \text{elmin}(\alpha_\ell^T, \mathbf{0}) \cdot (x_i - r \cdot \mathbf{1}) + \beta_\ell, 0) = 0,$$

¹Also there are several other verification tools based on abstract interpretation, such as CROWN [26] and DeepZ [27], which vary in precision, affecting loss function evaluations. We choose DeepPoly since it provides a good balance between efficiency and precision.

Algorithm 1 PATCHPRO

Input: Original DNN N , pairs of input and its adversarial examples $\{(x_i, x_i^*)\}_{i=1}^n$, and radius $r > 0$
Output: A repaired DNN $F = (N, \mathcal{C}, \mathcal{P}, \tau)$
1: $\mathcal{C} \leftarrow (B(x_1, r), \dots, B(x_n, r))$, iter $\leftarrow 0$
2: Init(\mathcal{P}) \triangleright Initialize the patch modules $\mathcal{P} = (P_1, \dots, P_n)$
3: $D[\cdot, \cdot] \leftarrow \{\}$ \triangleright A dictionary where $D[i]$ stores properties fixed with P_i
4: **for** $i \leftarrow 1$ to n **do**
5: $D[i] \leftarrow \{(F, B(x_i, r))\}$
6: $E \leftarrow \{1, \dots, n\}$ \triangleright Record the properties not fixed yet
7: **while** iter $< M$ **do** $\triangleright M$: Maximum number of iterations
8: iter \leftarrow iter + 1
9: **for** $j \in E$ **do**
10: $(P_j, T, \text{Repaired}) \leftarrow \text{TRAIN}(N, P_j, D[j])$ \triangleright Alg. 2
11: **if** Repaired = **True** **then**
12: $E \leftarrow E \setminus \{j\}$
13: **else**
14: **for** $\varphi \in T$ **do** \triangleright We write $\varphi = (F, X)$
15: **for** $d \leftarrow 1$ to n_0 **do**
16: score $_d \leftarrow \partial_d \mathcal{L}(\varphi)(x) \cdot \sup_{x, x' \in X} |x_d - x'_d|$
17: $d^* \leftarrow \arg \max_d \text{score}_d$
18: $X_1, X_2 \leftarrow \text{Bisect}(X, d^*)$ \triangleright Bisection on the d^* th dimension
19: $\varphi_1 \leftarrow (F, X_1)$, $\varphi_2 \leftarrow (F, X_2)$, $D[j] \leftarrow (D[j] \cup \{\varphi_1, \varphi_2\}) \setminus \{\varphi\}$
20: **if** $E = \{\}$ **then**
21: **return** F \triangleright Repair with provable guarantee
22: **return** F \triangleright Repair without provable guarantee

Algorithm 2 Patch training

Input: Original DNN N , DNN P as a patch module, and a finite set U of local robustness properties
Output: The optimized patch module P , the set $T \subseteq U$ of properties to be refined, and whether the properties in U have all been fixed with a provable guarantee
1: **function** TRAIN(N, P, U)
2: epoch $\leftarrow 0$
3: **while** epoch $< R$ **do** $\triangleright R$: maximum number of epochs
4: epoch \leftarrow epoch + 1
5: $w \leftarrow w - \eta \cdot \nabla \mathcal{L}^*(\bigwedge U)(w)$ $\triangleright w$: the weights in P
6: **if** $\mathcal{L}^*(\bigwedge U)(w) = 0$ **then** $\triangleright \mathcal{L}^*(\bigwedge U) := \sum_{\varphi \in U} \mathcal{L}^*(\varphi)$
7: **return** $(P, \{\}, \text{True})$
8: $T \leftarrow \text{Slice}_K(\text{ArgSort}_\varphi \{\mathcal{L}^*(\varphi)(w) \mid \mathcal{L}^*(\varphi)(w) > 0, \varphi \in U\})$
9: **return** (P, T, False)

where elmax and elmin are the element-wise max and min operation, $\mathbf{0}$ and $\mathbf{1}$ are the vector in \mathbb{R}^{n_0} with all the entries 0 and 1, respectively, then the property φ holds.

The notion that $\mathcal{L}^*(\varphi)$ is the expansion of $\mathcal{L}(\varphi)(x)$ after taking corresponding values on the boundary of the ball. To improve the precision of $\mathcal{L}^*(\varphi)$ obtained from DeepPoly, we employ the input interval partitioning technique from ART [25]. It selects the partition dimension by computing the multiplication of the partial derivative of the safety violated loss function $\mathcal{L}(\varphi)(x)$ and the size of the input interval in the corresponding dimension, and bisects the box region over the dimension with the maximum score. After partitioning, the property φ is split into two new properties, whose input sets are two sub-boxes of the original input region, and the union of these two sub-boxes is the input set of φ .

The main algorithm of PATCHPRO is shown in Alg. 1. We construct a dictionary D , where $D[i]$, initialized as $\{(F, B(x_i, r))\}$, stores the properties that the i th patch module

P_i repairs (Line 3–5, Alg. 1). The repair process has at most M iterations. In each iteration, we train the patches which do not yet provide provable repair, i.e., those whose index is in E . The algorithm of training a specific patch module P for repairing a set U of properties is shown in Alg. 2, which outputs the optimized patch module P , the set T of properties to be refined, and a boolean label recording whether the repair of the patch P has a provable guarantee. A standard gradient descent procedure is run until a provable repair is achieved, i.e., $\mathcal{L}^*(\bigwedge U)(w) = 0$, or the number of epochs reaches a threshold R (Line 3–7, Alg. 2). After that, if it is still not provable, we sort $\mathcal{L}^*(\varphi)(w)$ for $\varphi \in U$ from the largest to the smallest, and extract the largest K ones (whose value is strictly larger than 0) as the properties to be refined (Line 8, Alg. 2). For a property $\varphi = (F, X)$ to be refined, we select an input dimension to bisect the input space X . For an input dimension d , we define

$$\text{score}_d = \partial_d \mathcal{L}(\varphi)(x) \cdot \sup_{x, x' \in X} |x_d - x'_d|,$$

where $\partial_d \mathcal{L}(\varphi)(x)$ is the partial derivative of $\mathcal{L}(\varphi)(x)$ on the d th dimension. We choose the dimension with the largest score to bisect X , and the property $\varphi = (F, X)$ is refined to two properties $\varphi_1 = (F, X_1)$ and $\varphi_2 = (F, X_2)$, recorded in the dictionary D (Line 14–19, Alg. 1). At the end of each iteration, we check whether all the patches provide provable repair; if so, it terminates immediately and outputs the current repaired DNN F , and this repair is provable (Line 20–21, Alg. 1). If E is still non-empty after M iterations, it outputs F without provable guarantee (Line 22, Alg. 1). Although we are focusing on fixing adversarial attacks in this work, PATCHPRO also works for repair safety properties of DNNs by making minor changes in Alg. 1. One detailed example can be found in Supplementary Material [28].

In PATCHPRO, we follow a classical way of defining the loss function with the linear relaxation obtained from DeepPoly, and incorporate it into our patch-based repair framework. This combination has several advantages. First, every patch is responsible for repairing properties in a specific pattern, which makes it much easier to obtain a provable repair; even if the repair is not provable, the safety violated loss is significantly declining in the training process, and it is highly possible that the repaired DNN is locally robust. Different from the neuron-level repair, there is no modification on the original DNN N in PATCHPRO, and due to the design of the indicator, the patch modules do not affect the behaviors of other input before τ allocates a patch to it. This mechanism avoids the drawdown of accuracy, which is quite severe in many neuron-level repair methods. Also, by allocating patches with τ , we can achieve good generalization of our repair to other inputs, and it is the key to solving this essential challenge in fixing adversarial attacks. In the following, we will present how to construct the patch allocation function τ to improve generalization.

For a network N with L layers and a maximum of n neurons per layer, the time complexity of running DeepPoly for one neuron is $O(n^2 \cdot L)$ [23]. The time complexity of

running Algorithm 2 once, which involves backpropagation for gradient computation and update weights of the patch networks in line 5 of this algorithm, is $O(R \cdot n_{\text{patch}} \cdot L_{\text{patch}})$, where n_{patch} is the maximum number of neurons per layer in all patch networks, and L_{patch} is the number of layers in the patch networks. Furthermore, during the execution of PATCHPRO, difficult-to-repair properties are split into two easier-to-repair properties, which may increase the number of properties to repair up to a predefined upper limit, denoted as K . In summary, the overall time complexity of PATCHPRO is $O(R \cdot M \cdot K \cdot (n^3 \cdot L^2 + n_{\text{patch}}^3 \cdot L_{\text{patch}}^2))$.

C. Patch allocation

For generalization purposes, it is necessary to introduce a patch allocation function τ , given that the current patch construction approach lacks inherent adaptability to in-distribution data points. This patch allocation function τ is the key to achieving good generalization of the patch modules trained locally to global inputs.

To match the best patch modules for an input $x' \notin \bigcup_{i=1}^n B(x_i, r)$, we propose utilizing the prediction $\ell_0 = C_N(x')$ of the original network on x' as a guiding principle, with the aim of selecting patches that share the same ground truth label as ℓ_0 . Accordingly, we formally define the set of patch modules $\tau(x')$ associated with input x' as $\tau(x') = \{i \mid C_N(x') = C_N(x_i)\}$. This definition establishes that the set $\tau(x')$ encompasses all indices i satisfying the condition $C_N(x') = C_N(x_i)$, i.e., those instances where the prediction ℓ_0 made by the original network on x' is identical to the prediction made on sample x_i at index i . By determining the patch module set $\tau(x')$ in this manner, we ensure consistency between the selected modules and the input x' in terms of their predictions by the original network. This, in turn, is expected to enhance the robustness repair effectiveness for the specific input x' .

Once a set of patch modules has been allocated to an input x' , these modules effectively repair the entire robustness region of that input. Namely, we establish a new local robustness property such that any subsequent inputs $x \in B(x', r)$ will also utilize the same set of patch modules as x' , with the system's response given by $F(x) = \sum_{j \in \tau(x')} P_j(x) + N(x)$. In this context, the defense mechanism is established prior to the occurrence of adversarial attacks, obviating the need to re-allocate patch modules for each individual input x within the robustness region, which would otherwise require computing $\tau(x)$ separately. This proactive construction of the defense ensures a consistent and efficient protection strategy against potential adversarial threats across the entire neighborhood of x' , reinforcing the overall robustness.

We seemingly have problem when x' is not correctly classified by N , because in this case we may allocate inappropriate patches. If x' is an adversarial example, we can employ sampling-based methods like [29] to detect it and recognize its correct classification with a high probability. Otherwise, the wrong classification of x' may result from backdoor attacks,

biased training data, overfitting, etc, and such situations are beyond the scope of fixing adversarial attacks in this work.

D. Repair in a feature space

Repairing large DNNs with high-dimensional inputs poses significant challenges due to the substantial memory and computational costs associated with formal verification techniques. These costs can often become prohibitive, especially when dealing with deep networks where the precision of abstraction-based methods, such as DeepPoly, deteriorates significantly over multiple layers of propagation. Consequently, the key to adopting PATCHPRO to large DNNs is to reduce the input dimensionality and the size of the neural network in formal verification.

In the popular architectures of convolutional neural networks, the convolutional layers play the role of extracting important features from data, followed by several fully connected layers for classification according to these extracted features, so we call such a fully connected layer a feature space.

Definition 5. Let $N: \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_L}$ be a DNN where the function from the i th layer to the $(i+1)$ th is f_i , i.e., $N = f_{L-1} \circ \dots \circ f_0$. For $l \in [1, L]$, the feature space of the l th layer is \mathbb{R}^{n_l} , and the behavior of the input $x \in \mathbb{R}^{n_0}$ in this feature space is $N_{0,l} := f_{l-1} \circ \dots \circ f_0(x)$. The neural network starting with the feature space of the l th layer in N is $N_{l,L} = f_{L-1} \circ \dots \circ f_l$.

The dimensionality of a feature space is usually far smaller than that of the input layer. If we start with a feature space in employing DeepPoly, it does not need to calculate the abstraction of a great deal of convolutional layers, thus getting its efficiency and precision enhanced. Selecting the input feature space for repairing remains an open challenge, with limited theoretical research available. Our choice of the second-to-last fully connected layer is heuristic, based on the common understanding that earlier layers in a neural network are primarily responsible for feature extraction, while later layers handle decision-making. Repairing in the decision layer is more efficient and incurs lower abstraction costs.

To conduct repair in a given feature space with PATCHPRO, we first give an approximation of the feature space on every $B(x_i, r)$. Here we do not require that this approximation should be a sound abstraction of the real semantics of the feature space on $B(x_i, r)$, because calculating such an over-approximation with high precision is quite time-consuming. Instead, we simply sample in $B(x_i, r)$ and obtain the tightest box region that contains the buggy behaviors of the samples in the feature space as the approximation. To identify samples in the feature space that are “close” to adversarial examples, we use the Projected Gradient Descent (PGD) method and Fast Gradient Sign Method (FGSM) to find samples in $B(x_i, r)$, and obtain their corresponding feature space behaviors. Formally, the abstraction $(X_{(i)})_j$ can be defined as follows.

Definition 6. Let S be the set of samples obtained by applying PGD and FGSM to N in $B(x_i, r)$. For each sample $s \in S$, we

compute its feature space behavior $N_{0,l}(s)$ at the l -th layer. The abstraction of S in the feature space is defined as:

$$(X_{(i)})_j = [\min_{s \in S} (N_{0,l}(s))_j, \max_{s \in S} (N_{0,l}(s))_j]$$

where $(X_{(i)})_j$ is the range of j -th dimension of $X_{(i)}$, $(N_{0,l}(s))_j$ is the j -th element of the $N_{0,l}(s)$, and $j = 1, \dots, n_l$.

These samples, being actual adversarial examples in real-world scenarios, provide direct guidance for feature space repair. This approximation is not sound yet, but it does not mean that the inputs in $B(x_i, r)$ whose behaviors in the feature space are not within this approximation will not be repaired by the corresponding patch module P_i , because the indicator $\mathcal{I}_{\mathcal{C}}$ still remains the same. Since the approximation of the feature space is not sound, the repair does not have a provable guarantee. It is also worth mentioning that, even if two properties with different ground truth labels have their feature spaces overlapping, our repair still works. In this situation, although two contradicted properties are involved in the training process, but they must correspond to two different robustness regions $B(x_i, r)$, and thus different patch modules P_i . The correspondence of the robustness regions $B(x_i, r)$ and the patch P_i is always preserved, so P_i is always working for repairing on $B(x_i, r)$ with the correct classification label ℓ_0 .

After obtaining an approximation B_S for each $B(x_i, r)$ in the feature space, we employ Alg. 1, where the initialization of $D[i]$ in Line 5 is $(N_{l,L} + P_i, X_{(i)})$ instead. Since the repair in the feature space is not provable, the safety violation loss function $\mathcal{L}(\varphi)$ for $\varphi = (N_{l,L} + P_i, X_{(i)})$ in Def. 4 is modified to be

$$\mathcal{L}(\varphi)(x) = \sum_{\ell \neq \ell_0} (\alpha_{\ell}^T x + \beta_{\ell}),$$

where $\alpha_{\ell}^T x + \beta_{\ell}$ for $\ell \neq \ell_0$ is obtained by DeepPoly abstracting $N_{l,L} + P_i$ on $X_{(i)}$, and $\mathcal{L}^*(\varphi)$ is modified accordingly. Also, we do not run Line 6–7 in Alg. 2 and skip Line 6, 11–12 and 20–21 in Alg. 1, so that the loss \mathcal{L}^* decreases as much as possible and we can achieve a better repair performance.

In performing feature layer repairs on large-scale networks, we persist in utilizing the τ function guided by the original network output to allocate corresponding repair modules. While the approximation $X_{(i)}$ is indeed effective in capturing commonality in buggy behaviors, establishing a direct association between the input neighborhood $B(x', r)$ of a newly encountered point x' and its distribution within the feature layers of the extensive network proves challenging. Consequently, assigning repair modules based on the input neighborhood of x' emerges as a more targeted and reliable approach.

Repair in a feature space is an effective way to make PATCHPRO scale on large DNNs. Although we sacrifice provability, its repair performance and generalization capabilities demonstrate remarkable effectiveness in practical scenarios, which we will see in our experimental evaluation.

TABLE I

RESULTS OF REPAIRING LOCAL ROBUSTNESS. WE REPRESENT EACH TOOL'S NAME WITH THEIR FIRST THREE LETTERS, SUCH AS CAR FOR CARE, APR FOR APRNN, ETC. AND "—" MEANS TIMEOUT OR MEMORY OVERFLOW.

Model	r	n	RSR/%							DD/%							Time/s						
			CAR	PRD	APR	REA	TRA	ART	Ours	CAR	PRD	APR	REA	TRA	ART	Ours	CAR	PRD	APR	REA	TRA	ART	Ours
FNN_small	0.05	50	8.0	100.0	100.0	100.0	54.0	10.0	100.0	1.0	30.9	19.7	0.0	3.3	86.8	0.0	2.0	35.9	3.5	381.3	5.2	257.7	14.3
		100	2.0	100.0	100.0	100.0	2.0	10.0	100.0	0.1	25.8	29.6	0.0	86.8	87.0	0.0	2.3	80.8	9.6	762.5	13.4	318.5	24.8
		200	4.5	100.0	0.0	100.0	3.5	11.0	100.0	0.4	39.5	66.2	0.0	86.8	86.5	0.0	3.2	268.2	82.8	1486.6	24.6	422.3	60.6
		500	5.8	100.0	0.0	100.0	2.8	11.0	100.0	1.7	56.1	68.3	0.0	86.8	86.8	0.0	8.2	1358.4	73.0	3812.3	54.1	688.7	236.7
		1000	11.4	100.0	0.0	-	3.9	11.0	100.0	16.6	43.1	69.2	-	86.8	87.0	0.0	17.3	3473.5	34.3	-	103.0	1247.5	756.7
		50	2.0	100.0	100.0	100.0	8.0	14.0	100.0	0.0	71.3	50.9	0.0	86.8	86.5	0.0	2.2	51.6	3.8	358.8	8.2	393.0	15.5
	0.1	100	2.0	100.0	0.0	100.0	13.0	9.0	100.0	0.0	62.4	48.5	0.0	86.8	87.0	0.0	1.7	92.2	72.8	737.7	13.4	321.4	37.3
		200	1.0	100.0	0.0	100.0	10.5	12.0	100.0	0.3	60.1	63.9	0.0	86.8	86.5	0.0	2.8	321.8	149.2	1403.8	22.7	416.7	62.5
		500	0.4	100.0	0.0	100.0	10.0	14.0	100.0	-0.1	51.6	71.8	0.0	86.8	86.8	0.0	5.1	1324.6	27.9	3482.9	53.7	564.4	193.2
		1000	0.6	100.0	0.0	-	10.0	14.0	100.0	0.1	42.4	73.0	-	86.8	86.8	0.0	8.0	3844.8	33.9	-	104.5	1733.2	768.1
		50	0.0	100.0	100.0	100.0	8.0	8.0	100.0	0.0	77.0	42.9	0.0	86.8	86.8	0.0	1.8	55.3	3.6	328.3	13.3	255.7	18.6
		100	6.0	100.0	100.0	100.0	13.0	11.0	100.0	7.0	74.1	61.5	0.0	86.8	87.0	0.0	2.6	115.7	9.5	670.3	13.4	322.3	52.0
	0.3	200	0.0	100.0	0.0	100.0	10.5	8.0	100.0	0.0	62.8	54.2	0.0	86.8	87.0	0.0	3.2	364.9	132.1	1306.2	23.2	405.6	115.8
		500	0.0	100.0	0.0	100.0	10.0	10.0	100.0	-0.1	57.1	65.7	0.0	86.8	86.3	0.0	4.8	1934.4	147.9	3248.1	52.5	555.8	255.9
		1000	0.0	100.0	0.0	-	9.8	11.0	100.0	0.0	58.3	83.9	-	86.8	86.8	0.0	7.4	3759.6	44.7	-	103.2	2116.1	935.8
FNN_big	0.05	50	6.0	100.0	100.0	100.0	98.0	24.0	100.0	2.0	0.7	3.6	0.0	0.4	85.8	0.0	1.7	89.5	11.2	1807.6	5.7	436.5	14.5
		100	10.0	100.0	100.0	100.0	78.0	9.0	100.0	2.8	0.8	16.0	0.0	-0.1	86.9	0.0	3.6	171.1	24.3	3673.7	14.3	503.5	29.1
		200	10.0	100.0	100.0	100.0	73.0	6.0	100.0	4.8	1.0	28.1	0.0	0.1	88.3	0.0	4.5	419.0	92.3	7399.1	24.3	628.3	51.2
		500	10.0	100.0	100.0	-	67.0	10.0	100.0	6.1	1.6	25.8	-	-0.1	87.1	0.0	9.2	1390.1	657.6	-	54.4	1080.2	290.6
		1000	11.2	100.0	100.0	-	71.9	20.0	100.0	6.2	1.9	28.0	-	0.1	85.8	0.0	17.9	4139.0	5499.2	-	108.1	2167.6	1033.2
		50	18.0	100.0	100.0	100.0	8.0	8.0	100.0	13.7	6.7	13.9	0.0	0.8	87.4	0.0	2.7	88.2	10.5	1612.1	7.1	442.2	13.9
	0.1	100	2.0	100.0	100.0	100.0	27.0	11.0	100.0	0.1	5.6	25.8	0.0	8.8	87.4	0.0	2.9	173.7	22.8	3337.2	13.5	514.6	32.1
		200	17.5	100.0	100.0	100.0	15.5	10.0	100.0	10.6	6.1	27.7	0.0	0.7	87.4	0.0	6.4	445.5	95.8	6801.2	23.3	639.2	75.4
		500	0.6	100.0	100.0	-	15.4	15.0	100.0	1.3	7.4	33.7	-	0.5	88.3	0.0	6.3	1403.6	684.8	-	57.1	1048.6	356.8
		1000	0.5	100.0	100.0	-	26.4	11.0	100.0	0.7	8.3	33.3	-	0.5	86.9	0.0	11.2	4321.6	4726.4	-	107.8	2148.3	735.9
		50	0.0	100.0	100.0	100.0	0.0	14.0	100.0	0.0	27.5	33.3	0.0	0.5	85.8	0.0	1.8	119.5	11.6	1280.9	7.1	439.2	17.1
		100	1.0	100.0	100.0	100.0	4.0	14.0	100.0	0.0	31.8	56.6	0.0	1.8	85.8	0.0	2.1	253.1	27.0	2612.3	13.5	518.9	37.0
	0.3	200	1.0	100.0	100.0	100.0	2.0	10.0	100.0	0.2	28.5	28.4	0.0	0.4	86.9	0.0	3.7	630.5	105.6	5186.7	23.9	643.3	100.4
		500	0.4	100.0	100.0	-	2.2	9.0	100.0	0.2	23.9	29.5	-	0.6	86.9	0.0	4.9	2117.7	671.8	-	55.2	1047.6	365.4
		1000	0.6	100.0	100.0	-	4.8	12.0	100.0	0.7	23.6	26.3	-	0.8	85.8	0.0	10.9	5480.3	4297.3	-	108.8	2158.5	1246.4
CNN	0.05	50	0.0	100.0	100.0	100.0	100.0	0.0	100.0	2.4	0.8	0.0	0.0	0.9	88.5	0.0	3.5	6.2	57.9	509.3	4.3	789.9	14.1
		100	0.0	100.0	100.0	100.0	92.0	60.0	100.0	2.4	0.7	0.1	0.0	0.2	88.5	0.0	6.0	5.1	99.0	910.7	10.3	879.0	23.4
		200	0.0	100.0	100.0	100.0	88.0	0.0	100.0	2.7	1.0	0.0	0.0	0.0	88.5	0.0	9.8	19.0	182.9	1825.6	18.0	969.9	83.6
		500	0.0	100.0	100.0	-	92.2	-	100.0	3.4	1.2	-0.1	-	0.0	-	0.0	26.9	146.2	547.8	-	40.4	-	221.9
		1000	0.0	100.0	100.0	-	91.2	-	100.0	7.5	1.8	0.0	-	0.0	-	0.0	38.1	549.1	1611.6	-	80.9	-	1001.9
		50	0.0	100.0	100.0	100.0	94.0	0.0	100.0	0.6	1.0	0.1	0.0	0.4	88.0	0.0	2.6	45.6	56.6	456.0	5.7	938.0	12.7
	0.1	100	0.0	100.0	100.0	100.0	94.0	10.0	100.0	1.3	1.0	0.1	0.0	0.7	88.7	0.0	5.1	9.8	104.8	896.8	10.4	1022.8	29.5
		200	0.0	100.0	100.0	100.0	91.5	20.0	100.0	2.8	2.2	0.3	0.0	0.4	88.5	0.0	8.1	41.1	178.5	1800.8	17.7	1108.1	75.1
		500	0.0	100.0	100.0	-	91.6	-	100.0	9.1	1.7	0.4	-	0.5	-	0.0	24.9	139.2	458.5	-	40.6	-	337.0
		1000	0.0	100.0	100.0	-	92.8	-	100.0	5.4	2.4	0.2	-	0.5	-	0.0	32.0	492.7	1526.0	-	80.4	-	700.2
		50	0.0	100.0	100.0	100.0	0.0	10.0	100.0	0.0	43.2	7.2	0.0	0.8	88.5	0.0	2.2	3.8	59.9	453.8	5.6	808.2	15.0
		100	0.0	100.0	100.0	100.0	0.0	10.0	100.0	0.0	38.8	5.0	0.0	0.2	89.3	0.0	2.6	29.4	105.4	907.5	10.1	899.5	32.6
	0.3	200	0.0	100.0	100.0	100.0	0.0	0.0	100.0	0.0	36.8	3.6	0.0	0.2	89.3	0.0	4.6	59.9	196.2	1793.7	17.8	1014.7	87.9
		500	0.0	100.0	100.0	-	0.2	-	100.0	0.5	48.8	3.2	-	0.3	-	0.0	9.2	167.9	518.8	-	39.9	-	361.4
		1000	0.0	100.0	100.0	-	0.6	-	100.0	0.1	56.5	2.2	-	0.4	-	0.0	16.8	423.8	1667.0	-	80.4	-	1282.6

IV. EXPERIMENTAL EVALUATION

In this section, we evaluate PATCHPRO by answering the following research questions:

RQ1: What is the overall performance of PATCHPRO in repairing local robustness and correcting safety property violations?

RQ2: Does the repaired DNN exhibit the capability to defend against new adversarial attacks?

RQ3: Does PATCHPRO have scalability to repair large networks?

RQ4: In the context of PATCHPRO, how does the size and quantity of patch modules influence the efficacy of the repair?

A. Setup

All the experiments are conducted on a machine with an AMD EPYC 7763 64-Core Processor, 256 GB of memory, and an NVIDIA GeForce RTX 3090 with 24 GB of GPU memory. Each experiment has a timeout set to 10 000 seconds.

a) Dataset: We conduct evaluations on four common datasets: MNIST [30], CIFAR-10 [31], Tiny ImageNet [32], and ACAS Xu [33], [34]. The first three are widely recognized benchmarks in the field for studying neural network robustness [4], [35], and ACAS Xu is a commonly used benchmark in research on neural network verification and repair [20], [22], [36]. On MNIST, we use two fully connected networks and one convolutional network, while on CIFAR-10, we train a VGG19 [37] and a ResNet18 [38]. Network architectures are detailed in supplementary material [28]. On Tiny ImageNet,

we train a Resnet152 and a WRN101-2 [39]. We assess our approach on 35 ACAS Xu DNNs, which, as documented in [36], are expected to satisfy Property-2 but exhibit violations. For the local robustness repair task, we generate adversarial samples using PGD [4] for DNNs trained on MNIST and CIFAR-10 (50, 100, 200, 500, and 1,000 samples), as well as Tiny ImageNet (500 and 1,000 samples). The radius r is set to 0.05, 0.1, and 0.3 for MNIST, $\frac{4}{255}$ and $\frac{8}{255}$ for CIFAR-10, and $\frac{2}{255}$ and $\frac{4}{255}$ for Tiny ImageNet. On ACAS Xu, we aim to repair the violation of Property-2 using one patch module while preserving the original performance. Although PATCHPRO repairs safety properties without requiring sample information, due to the needs of other tools such as CARE, we sample 500 counterexamples as the faulty inputs for repair. For CIFAR-10, the accuracies of VGG19 and ResNet18 are 93.4% and 88.3%, respectively. For Tiny ImageNet, the accuracies of WRN101-2 and ResNet152 are 64.4% and 68.2%, respectively. The maximum number of iterations M in Alg. 1 is set to 25. In Alg. 2, the maximum number of epochs R , learning rate η , and selection number K are set to 10, 10, and 800, respectively. All patch modules consist of a single linear layer, unless otherwise specified. On MNIST, the patch module takes the sample itself as input, while on CIFAR-10 and Tiny ImageNet, it takes the output from the network's penultimate layer as input. For sampling the feature space, we use PGD attack with a step size of $\frac{2}{255}$ for 10 rounds, with each round consisting of 50 steps. We collect all adversarial

TABLE II
RESULTS OF CORRECTING VIOLATION OF SAFETY PROPERTIES ON ACAS Xu

Model	RSR/%					RGR/%					FDD/%					Time/s				
	CAR	PRD	REA	ART	Ours	CAR	PRD	REA	ART	Ours	CAR	PRD	REA	ART	Ours	CAR	PRD	REA	ART	Ours
$N_{2,1}$	59.2	100.0	100.0	100.0	100.0	60.1	97.5	100.0	100.0	100.0	39.1	50.3	64.2	9.2	0.0	28.1	4.3	572.3	18.3	25.3
$N_{2,2}$	67.0	100.0	100.0	100.0	100.0	65.6	99.0	99.7	100.0	100.0	56.5	87.6	96.5	9.9	0.0	23.6	3.8	545.6	18.6	18.8
$N_{2,3}$	100.0	100.0	100.0	100.0	100.0	99.9	99.6	100.0	100.0	100.0	11.1	92.0	80.1	9.3	0.0	14.2	2.9	522.8	17.8	18.2
$N_{2,4}$	100.0	100.0	100.0	100.0	100.0	98.8	99.9	100.0	100.0	100.0	10.5	97.6	67.6	8.8	0.0	13.9	3.1	515.4	17.9	18.5
$N_{2,5}$	98.4	100.0	100.0	100.0	100.0	97.8	98.9	64.1	100.0	100.0	9.6	90.9	91.9	5.4	0.0	28.8	3.7	522.0	16.8	19.0
$N_{2,6}$	59.4	100.0	100.0	100.0	100.0	54.7	98.5	87.3	100.0	100.0	5.1	92.8	94.6	2.7	0.0	29.2	4.4	518.1	17.2	24.4
$N_{2,7}$	100.0	100.0	100.0	100.0	100.0	100.0	99.3	92.5	100.0	100.0	5.9	97.0	88.0	1.7	0.0	11.7	5.0	520.6	16.1	30.4
$N_{2,8}$	100.0	100.0	100.0	100.0	100.0	100.0	97.8	93.7	100.0	100.0	6.3	95.7	86.1	2.1	0.0	12.7	4.4	526.1	15.8	18.1
$N_{2,9}$	97.8	100.0	100.0	100.0	100.0	98.1	90.5	92.6	100.0	100.0	13.5	94.9	88.9	1.5	0.0	24.4	5.1	512.2	15.7	24.2
$N_{3,1}$	100.0	100.0	100.0	100.0	100.0	100.0	97.8	99.2	100.0	100.0	8.7	90.9	79.2	7.3	0.0	12.7	6.3	521.1	17.2	18.5
$N_{3,2}$	100.0	100.0	100.0	100.0	100.0	100.0	99.3	95.3	100.0	100.0	8.9	12.2	48.6	8.6	0.0	12.2	3.0	510.9	16.7	18.3
$N_{3,3}$	100.0	100.0	100.0	100.0	100.0	99.6	94.3	98.8	100.0	100.0	35.7	66.4	68.6	8.3	0.0	24.6	4.3	516.6	17.4	18.5
$N_{3,4}$	100.0	100.0	100.0	100.0	100.0	100.0	98.1	97.7	100.0	100.0	18.9	97.5	96.0	9.6	0.0	23.3	3.1	513.9	15.6	23.5
$N_{3,5}$	99.6	100.0	100.0	100.0	100.0	99.4	97.7	80.7	100.0	100.0	13.0	98.8	95.4	5.8	0.0	33.8	3.6	528.0	17.5	17.8
$N_{3,6}$	100.0	100.0	100.0	100.0	100.0	99.9	98.7	97.2	100.0	100.0	9.8	94.3	96.6	2.7	0.0	16.1	4.6	639.5	16.8	17.5
$N_{3,7}$	100.0	100.0	100.0	100.0	100.0	100.0	91.6	81.8	100.0	100.0	5.5	96.4	80.0	2.2	0.0	20.6	5.0	515.4	16.9	18.6
$N_{3,8}$	100.0	100.0	100.0	100.0	100.0	100.0	98.8	98.3	100.0	100.0	5.0	97.5	88.8	1.6	0.0	15.2	4.1	579.9	17.9	24.8
$N_{3,9}$	100.0	100.0	100.0	100.0	100.0	99.6	96.0	49.2	100.0	100.0	9.2	97.0	86.1	1.9	0.0	16.2	4.9	774.4	16.8	19.0
$N_{4,1}$	100.0	100.0	100.0	100.0	100.0	99.9	98.3	98.8	100.0	100.0	40.1	96.2	93.2	7.8	0.0	14.5	3.5	549.4	17.6	24.5
$N_{4,3}$	98.4	100.0	100.0	100.0	100.0	97.5	99.4	99.8	100.0	100.0	20.1	91.1	87.5	7.0	0.0	28.0	2.9	506.7	17.1	17.7
$N_{4,4}$	100.0	100.0	100.0	100.0	100.0	100.0	98.7	99.9	100.0	100.0	4.3	82.0	85.2	8.4	0.0	9.5	3.1	535.8	16.9	17.9
$N_{4,5}$	100.0	100.0	100.0	100.0	100.0	100.0	99.8	94.6	100.0	100.0	33.6	46.4	94.8	6.0	0.0	9.8	5.0	544.8	18.2	17.7
$N_{4,6}$	100.0	100.0	100.0	100.0	100.0	100.0	98.2	97.9	100.0	100.0	11.2	89.6	86.0	2.6	0.0	19.9	5.4	548.2	15.9	24.1
$N_{4,7}$	77.6	100.0	100.0	100.0	100.0	78.1	98.8	85.6	100.0	100.0	3.9	96.3	96.2	1.6	0.0	38.4	4.5	544.0	16.0	24.5
$N_{4,8}$	99.8	100.0	100.0	100.0	100.0	99.7	97.4	77.1	100.0	100.0	4.7	97.9	88.2	1.6	0.0	36.0	4.9	542.7	14.8	30.2
$N_{4,9}$	99.8	100.0	100.0	100.0	100.0	99.8	95.0	64.0	100.0	100.0	7.7	98.4	90.1	3.3	0.0	21.2	5.0	545.8	16.4	23.8
$N_{5,1}$	100.0	100.0	100.0	100.0	100.0	100.0	90.5	97.6	100.0	100.0	14.0	98.4	96.9	6.9	0.0	19.8	3.5	541.3	17.1	18.6
$N_{5,2}$	100.0	100.0	100.0	100.0	100.0	99.9	99.3	100.0	100.0	100.0	4.1	91.8	94.1	7.2	0.0	13.3	4.0	543.4	17.2	18.3
$N_{5,3}$	100.0	100.0	100.0	100.0	100.0	99.9	88.6	99.7	100.0	100.0	8.8	88.3	94.8	6.0	0.0	17.4	3.5	542.9	16.8	18.7
$N_{5,4}$	99.6	100.0	100.0	100.0	100.0	98.9	95.9	99.5	100.0	100.0	17.7	98.0	91.9	7.1	0.0	16.9	2.9	540.7	16.1	18.2
$N_{5,5}$	100.0	100.0	100.0	100.0	100.0	100.0	97.0	99.7	100.0	100.0	5.4	94.8	94.4	3.9	0.0	8.6	3.4	518.9	15.9	18.3
$N_{5,6}$	100.0	100.0	100.0	100.0	100.0	100.0	99.2	98.8	100.0	100.0	6.0	95.7	93.7	9.6	0.0	13.1	4.7	519.5	15.9	30.0
$N_{5,7}$	97.2	100.0	100.0	100.0	100.0	97.7	99.1	98.6	100.0	100.0	4.9	95.3	87.7	1.8	0.0	21.2	5.0	517.3	16.3	24.2
$N_{5,8}$	99.2	100.0	100.0	100.0	100.0	98.9	99.7	95.5	100.0	100.0	5.2	99.2	80.0	1.8	0.0	28.3	4.3	517.9	15.9	24.1
$N_{5,9}$	100.0	100.0	100.0	100.0	100.0	100.0	96.9	91.7	100.0	100.0	5.4	97.2	27.0	2.0	0.0	22.6	4.8	515.8	15.4	24.3
Avg	95.8	100.0	100.0	100.0	100.0	95.5	97.3	92.2	100.0	100.0	13.4	88.8	85.1	5.2	0.0	20.0	4.2	540.9	16.8	21.4

examples generated during this process. Additionally, we use FGSM attack to generate 50 adversarial examples, which are then combined with the samples obtained from PGD. This combined set of samples is used for sampling the feature space.

b) Baselines: We compare PATCHPRO with the state-of-the-art repair methods including CARE [8], APRNN [22], PRDNN [20], REASSURE [21] and ART [25]. Since APRNN and PRDNN require selecting a layer to repair on, we traverse all eligible layers across each baseline, and report the layer that exhibit the best performance. Additionally, for the local robustness repair task, we compare our approach with an adversarial training method TRADES [40], where we select the trained model with the best performance in 200 epochs with their default parameters.

c) Metrics: To assess generalization, we sample 10 adversarial examples on each $B(x_i, r)$ different from x_i^* to form a generalization set D_g for MNIST, CIFAR-10 and Tiny ImageNet, and use an independent set of 5 000 counterexamples for ACAS Xu. Besides these, we have an independent test set D_t of size 10 000, 10 000, 10 000 and 5 000 for MNIST, CIFAR-10, Tiny ImageNet and ACAS Xu, respectively. We employ AutoAttack [5] to attack the repaired DNN F on $\{x_i\}_{i=1}^n$ and D_t with the same radius r . The metrics we use include repair success rate (RSR), repair generalization rate (RGR), drawdown (DD), defense success rate (DSR), and defense generalization success rate (DGSR), defined as:

$$\begin{aligned}
\text{RSR} &= \frac{|\{i \mid C_F(x_i^*) = C_N(x_i)\}|}{n}, \\
\text{RGR} &= \frac{|\{x \in D_g \mid C_F(x) = \ell_x\}|}{|D_g|}, \\
\text{DD} &= \frac{|\{x \in D_t \mid C_N(x) = \ell_x\}| - |\{x \in D_t \mid C_F(x) = \ell_x\}|}{|D_t|}, \\
\text{DSR} &= \frac{|\{i \mid \forall x \in \text{AA}(F, B(x_i, r)), C_F(x) = C_N(x_i)\}|}{n}, \\
\text{DGSR} &= \frac{|\{x \in D_t \mid \forall x' \in \text{AA}(F, B(x, r)), C_F(x') = C_N(x)\}|}{|D_t|},
\end{aligned}$$

where ℓ_x denotes the ground truth of x , and $\text{AA}(\varphi)$ represents the set of potential adversarial samples obtained by attacking the local robustness property φ with AutoAttack. On ACAS Xu, the metric Drawdown is replaced with Fidelity Drawdown (FDD) because there is no labelled ground truth for inputs on ACAS Xu:

$$\text{FDD} = \frac{|\{x \in D_t \mid C_F(x) \neq C_N(x)\}|}{|D_t|}.$$

The metrics RSR and DD (or FDD) evaluates the overall performance of DNN repair by measuring the percentage of buggy inputs successfully repaired and how much the accuracy decreases, while RGR, DSR and DGSR reflects how the repair generalizes to the robustness regions $B(x_i, r)$ and to other inputs.

B. Repair performance

The results of the local robustness repair task on MNIST are summarized in Table I. We evaluate overall repair per-

TABLE III
RESULTS OF GENERALIZATION AND DEFENSE AGAINST NEW ADVERSARIAL ATTACKS

Model	r	n	RGR/%							DSR/%							DGSR/%						
			CAR	PRD	APR	REA	TRA	ART	Ours	CAR	PRD	APR	REA	TRA	ART	Ours	CAR	PRD	APR	REA	TRA	ART	Ours
FNN_small	0.05	50	7.6	44.4	47.2	10.8	80.0	10.0	100.0	0.0	0.0	2.0	0.0	26.0	6.0	100.0	8.9	3.2	2.3	0.0	9.4	8.9	91.3
		100	2.2	48.7	46.7	9.1	0.0	10.0	100.0	1.0	0.0	0.0	0.0	2.0	20.0	100.0	9.3	5.1	0.8	0.0	9.8	11.3	96.6
		200	5.0	42.6	16.1	8.5	0.0	11.0	100.0	2.0	0.0	0.5	0.0	3.5	10.0	100.0	8.6	1.1	1.3	0.0	9.8	10.1	96.6
		500	6.6	39.0	10.3	7.2	0.0	11.0	100.0	1.4	0.0	3.8	0.0	2.8	9.6	100.0	8.3	0.5	5.3	0.0	9.8	8.9	96.6
		1000	12.7	41.3	10.9	—	2.0	11.0	100.0	2.3	0.0	0.4	—	3.9	14.4	100.0	6.1	0.2	0.9	—	9.8	11.3	96.6
		50	0.8	24.2	32.0	2.4	20.0	14.0	100.0	0.0	0.0	0.0	0.0	0.0	8.0	100.0	0.0	0.0	0.0	0.0	9.8	9.6	96.3
	0.1	100	1.7	29.8	21.9	1.8	10.0	9.0	100.0	0.0	0.0	0.0	0.0	13.0	9.0	100.0	0.0	0.0	0.0	0.0	9.8	9.6	95.0
		200	1.1	32.7	12.9	1.4	5.0	12.0	100.0	0.0	0.0	0.0	0.0	10.5	11.5	100.0	0.0	0.0	0.0	0.0	9.8	10.1	95.2
		500	1.7	37.5	17.3	1.4	8.0	14.0	100.0	0.0	0.0	0.0	0.0	10.0	10.0	100.0	0.0	0.0	0.1	0.0	9.8	9.8	95.3
		1000	2.3	42.5	15.2	—	13.0	14.0	100.0	0.0	0.0	1.2	—	10.0	10.0	100.0	0.0	0.0	0.2	—	9.8	9.8	96.5
		50	0.0	24.2	37.4	0.8	20.0	8.0	100.0	0.0	0.0	0.0	0.0	8.0	8.0	100.0	0.0	0.0	0.0	0.0	9.8	9.7	96.5
		100	7.7	22.0	18.4	0.4	10.0	11.0	100.0	0.0	0.0	0.0	0.0	13.0	11.0	100.0	0.0	0.0	0.0	0.0	9.8	9.6	96.5
	0.3	200	0.0	26.1	8.8	0.5	5.0	8.0	100.0	0.0	0.0	0.0	0.0	10.5	9.5	100.0	0.0	0.0	0.0	0.0	9.8	9.6	96.6
		500	0.4	30.4	12.8	0.3	8.0	10.0	100.0	0.0	0.0	0.0	0.0	10.0	10.4	100.0	0.0	0.0	0.0	0.0	9.8	10.3	96.6
		1000	0.6	31.9	12.5	—	13.0	11.0	100.0	0.0	0.0	0.0	—	9.8	9.8	100.0	0.0	0.0	0.0	—	9.8	9.8	96.6
FNN_big	0.05	50	6.6	63.0	64.8	8.4	100.0	24.0	100.0	2.0	2.0	8.0	0.0	82.0	24.0	100.0	37.7	38.3	31.7	0.0	53.5	11.3	97.2
		100	10.9	63.3	67.4	9.6	90.0	9.0	100.0	7.0	1.0	11.0	0.0	60.0	9.0	100.0	35.7	39.8	19.0	0.0	54.9	10.3	97.2
		200	10.7	58.9	64.6	7.3	69.5	6.0	100.0	4.5	2.5	2.5	0.0	54.5	5.0	100.0	37.9	33.2	12.8	0.0	57.0	8.9	97.2
		500	10.6	54.8	66.2	—	78.8	10.0	100.0	4.8	0.6	0.0	—	52.8	7.4	100.0	35.4	29.7	1.7	—	61.2	10.1	97.2
		1000	11.8	54.6	68.4	—	79.2	20.0	100.0	7.1	0.4	0.0	—	56.1	19.2	100.0	36.8	25.5	1.8	—	66.2	11.3	97.2
		50	20.2	52.6	44.2	3.4	0.0	8.0	100.0	2.0	0.0	0.0	0.0	0.0	8.0	100.0	2.6	0.7	1.1	0.0	1.8	9.7	97.2
	0.1	100	2.0	47.5	46.3	3.2	20.0	11.0	100.0	0.0	0.0	0.0	0.0	7.0	11.0	100.0	1.8	0.9	0.0	0.0	4.0	9.8	97.2
		200	19.2	49.6	46.8	2.5	22.5	10.0	100.0	1.5	0.0	0.0	0.0	6.0	10.5	100.0	2.6	0.8	0.1	0.0	4.9	9.8	97.2
		500	1.2	50.6	49.3	—	23.6	15.0	100.0	0.0	0.2	0.0	—	2.8	9.0	100.0	1.2	0.2	0.0	—	5.4	8.9	97.2
		1000	1.0	51.1	49.9	—	42.1	11.0	100.0	0.0	0.0	0.0	—	7.2	10.5	100.0	1.4	0.1	0.0	—	9.2	10.3	97.2
		50	0.0	42.2	20.4	2.8	2.0	14.0	100.0	0.0	0.0	0.0	0.0	0.0	14.0	100.0	0.0	0.0	0.0	0.0	0.0	11.3	96.5
		100	0.1	43.4	27.8	1.4	2.0	14.0	100.0	0.0	0.0	0.0	0.0	0.0	14.0	100.0	0.0	0.0	0.0	0.0	0.0	11.3	96.5
	0.3	200	0.1	40.8	31.0	1.6	11.0	10.0	100.0	0.0	0.0	0.0	0.0	0.0	10.0	100.0	0.0	0.0	0.0	0.0	0.0	10.3	96.6
		500	0.3	36.9	36.0	—	9.2	9.0	100.0	0.0	0.0	0.0	—	0.0	10.2	100.0	0.0	0.0	0.0	—	0.0	10.3	97.2
		1000	0.4	36.5	37.1	—	15.0	12.0	100.0	0.0	0.0	0.0	—	0.0	11.8	100.0	0.0	0.0	0.0	—	0.0	11.3	97.2
CNN	0.05	50	27.0	69.6	67.6	47.2	100.0	0.0	100.0	0.0	0.0	12.0	0.0	100.0	18.0	100.0	70.6	37.5	70.4	0.0	87.0	9.8	96.6
		100	29.4	74.5	65.6	45.7	100.0	60.0	100.0	1.0	0.0	10.0	0.0	86.0	27.0	100.0	70.6	29.4	69.2	0.0	88.2	9.7	98.3
		200	27.6	79.8	62.5	50.9	97.0	0.0	100.0	1.0	0.0	3.0	0.0	84.0	20.5	100.0	70.6	10.4	70.6	0.0	89.9	9.7	98.3
		500	30.8	89.0	62.3	—	92.4	—	100.0	1.4	0.0	0.8	—	87.0	—	100.0	70.6	4.3	71.4	—	91.2	—	98.3
		1000	32.9	92.2	63.8	—	90.8	—	100.0	1.5	0.0	0.2	—	86.8	—	100.0	70.7	0.9	65.7	—	92.1	—	98.3
		50	12.2	53.0	41.0	10.4	80.0	0.0	100.0	2.0	0.0	2.0	0.0	50.0	8.0	100.0	8.3	0.0	8.7	0.0	44.9	10.3	98.2
	0.1	100	22.2	69.1	46.0	14.7	88.0	10.0	100.0	1.0	0.0	0.0	0.0	57.0	11.0	100.0	8.2	0.0	8.6	0.0	48.5	9.6	98.2
		200	21.8	73.2	46.1	13.0	96.5	20.0	100.0	0.5	0.0	0.5	0.0	62.0	11.0	100.0	8.2	0.0	9.2	0.0	53.3	9.8	98.2
		500	30.4	84.5	47.0	—	93.2	—	100.0	0.2	0.0	0.0	—	66.6	—	100.0	8.2	0.0	8.4	—	62.4	—	98.2
		1000	24.6	90.3	48.7	—	94.7	—	100.0	0.6	0.0	0.0	—	70.1	—	100.0	8.3	0.0	4.6	—	67.8	—	98.3
		50	0.0	44.8	70.0	4.0	0.0	10.0	100.0	0.0	0.0	0.0	0.0	0.0	12.0	100.0	0.0	0.0	0.0	0.0	0.0	9.8	98.3
		100	0.0	59.9	75.5	2.7	13.0	10.0	100.0	0.0	0.0	0.0	0.0	0.0	7.0	100.0	0.0	0.0	0.0	0.0	0.0	8.9	98.3
	0.3	200	0.0	69.4	78.2	3.6	12.5	0.0	100.0	0.0	0.0	0.0	0.0	0.0	7.0	100.0	0.0	0.0	0.0	0.0	0.0	8.9	98.3
		500	1.3	79.6	73.3	—	12.6	—	100.0	0.0	0.0	0.0	—	0.0	—	100.0	0.0	0.0	0.0	—	0.0	—	98.3
		1000	1.1	85.9	70.6	—	16.0	—	100.0	0.0	0.0	0.0	—	0.0	—	100.0	0.0	0.0	0.0	—	0.0	—	98.2

formance with RSR, DD, and runtime, representing fix success, accuracy retention, and speed, respectively. PRDNN, REASSURE, and PATCHPRO reach 100% RSR due to their provable designs. APRNN, despite being provable, underperforms on some FNN_small cases, while ART struggles with high-dimensional inputs. CARE and TRADES, lacking provable guarantees, show inferior RSR compared to provable methods. CARE’s suboptimal repair across all models may stem from the complex interplay between local robustness repair and DNNs’ numerous parameters, making it difficult to pinpoint and modify relevant parts. Although TRADES performs better on CNNs, its effectiveness wanes with larger-radius local robustness errors. In terms of DD, PATCHPRO and REASSURE consistently outperform alternative methods across all models.

The results of correcting safety property violations on ACAS Xu are presented in Table II. Each patch module here is of the same size as the original network. PATCHPRO, PRDNN, REASSURE, and ART achieve a perfect 100% RSR with provable repairs. Both ART and our method showcase remarkable RGR scores, achieving a perfect 100% among all the evaluated tools. PATCHPRO also excels with a FDD of zero, outperforming CARE, ART, PRDNN, and REASSURE. These results emphasize our method’s strength in delivering provable repairs and preserving high-fidelity functionality.

C. Generalization

In the experiment, we assess the repaired network’s generalization and resistance to new adversarial attacks (Table III). PATCHPRO leads in RGR, demonstrating superior performance compared to other baselines. PRDNN, APRNN, and TRADES also show some degree of generalization, particularly on CNNs. For DSR, PATCHPRO outperforms competitors, with TRADES showing moderate defense against small-radius attacks on FNN_big and CNN. Evaluating DGSR, PATCHPRO again tops the list, while CARE, PRDNN, APRNN, and TRADES show limited defense against FNN_big and CNN. DGSR is key for gauging generalization to global inputs. Overall, these findings underscore PATCHPRO’ breakthrough in repairing adversarial attacks.

Answer to RQ1 and RQ2: PATCHPRO consistently outperforms the baselines in both local robustness repair and safety property violation correction tasks. It achieves 100% repair success rate coupled with 0% drawdown and acceptable efficiency. Additionally, it demonstrates significant generalization against unforeseen adversarial attacks.

D. Scalability Evaluation

We examine PATCHPRO’ scalability on VGG19 and ResNet18 for CIFAR-10 local robustness repair via feature-space repair. Results in Table IV show PATCHPRO, PRDNN, and APRNN achieve 100% RSR, while TRADES follows

TABLE IV
RESULTS OF REPAIRING LOCAL ROBUSTNESS ON CIFAR-10

		VGG19										ResNet-18									
		$r = 4/255$					$r = 8/255$					$r = 4/255$					$r = 8/255$				
		50	100	200	500	1000	50	100	200	500	1000	50	100	200	500	1000	50	100	200	500	1000
RSR/%	CARE	8.0	5.0	4.0	11.0	10.9	2.0	4.0	2.0	2.2	2.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	PRDNN	100.0	100.0	100.0	—	—	100.0	100.0	100.0	—	—	100.0	100.0	100.0	—	—	100.0	100.0	100.0	—	—
	TRADE	98.0	95.0	97.0	95.0	96.4	100.0	97.0	93.5	95.6	95.8	100.0	99.0	100.0	99.8	99.5	100.0	100.0	100.0	99.8	96.8
	APRNN	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	Ours	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
RGR/%	CARE	6.6	4.2	3.6	11.7	11.8	2.0	3.6	2.7	2.4	2.9	2.6	2.7	9.1	12.2	6.7	3.8	2.6	2.3	2.7	2.3
	PRDNN	69.8	68.2	68.5	—	—	58.0	66.6	61.4	—	—	68.0	68.6	67.2	—	—	59.8	57.8	54.9	—	—
	TRADE	100.0	86.0	90.0	97.6	97.0	100.0	100.0	95.0	98.0	98.9	100.0	100.0	100.0	100.0	99.1	100.0	100.0	100.0	100.0	97.3
	APRNN	69.2	68.9	70.9	75.5	80.4	63.2	65.2	68.2	66.1	70.7	70.6	65.1	62.0	62.7	68.2	65.0	55.6	53.0	49.6	59.0
	Ours	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
DD/%	CARE	0.1	0.2	0.2	0.2	0.3	0.1	0.2	0.2	0.1	0.3	0.1	0.1	0.2	0.3	0.2	0.1	0.0	0.0	0.1	0.1
	PRDNN	13.6	24.8	30.3	—	—	19.8	40.5	50.2	—	—	2.1	2.1	2.0	—	—	14.0	13.7	12.2	—	—
	TRADE	18.6	11.9	7.7	6.3	6.1	25.8	18.5	11.5	8.5	8.3	43.4	29.2	20.1	14.4	10.1	38.2	34.9	15.9	11.9	10.1
	APRNN	14.2	20.0	29.3	48.7	46.4	24.0	39.5	61.9	75.5	82.1	1.8	1.8	2.2	3.1	6.1	5.5	4.2	7.6	12.7	19.5
	Ours	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DSR/%	CARE	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	PRDNN	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	APRNN	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	TRADE	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Ours	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
DGSR/%	CARE	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	PRDNN	0.0	0.0	0.0	—	—	0.0	0.0	0.0	—	—	0.0	0.0	0.0	—	—	0.0	0.0	0.0	—	—
	APRNN	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	TRADE	3.6	3.7	2.9	3.4	3.4	4.1	3.7	3.2	2.3	2.6	5.7	6.7	6.0	5.4	5.2	5.6	5.0	5.4	5.8	5.2
	Ours	93.4	93.4	93.4	93.4	93.4	93.4	93.4	93.4	93.4	93.4	88.3	88.3	88.3	88.3	88.3	88.3	88.3	88.3	88.3	88.3
Time/s	CARE	47.8	64.4	106.5	380.4	928.2	36.9	48.2	96.0	234.1	443.0	45.5	58.9	105.5	243.5	384.9	37.1	75.4	101.6	180.7	285.7
	PRDNN	1731.8	2640.6	6307.4	—	—	1160.4	2210.2	5126.9	—	—	731.8	1794.9	3307.5	—	—	715.4	1300.2	5757.3	—	—
	TRADE	68.7	114.8	194.6	415.4	814.6	67.2	111.1	187.2	417.6	806.8	57.1	107.8	186.7	405.0	794.3	57.9	107.8	184.1	408.1	790.6
	APRNN	188.9	288.2	451.3	1645.6	6211.4	209.2	318.6	492.4	1750.4	7787.1	307.7	746.1	812.2	1338.1	1710.8	580.9	990.8	863.2	1556.0	1988.4
	Ours	238.7	471.6	963.0	2516.9	5441.6	229.9	470.7	950.1	2485.0	5436.0	265.8	525.1	1050.3	2586.4	5262.3	264.7	523.1	1042.8	2627.8	5325.1

closely. PATCHPRO leads in RGR and DD with TRADES following suit. PATCHPRO also significantly outperforms others in DSR and matches *accuracy levels* in DGSR, surpassing all tools. On larger datasets like Tiny ImageNet, PATCHPRO maintains superiority in repairing WRN101-2 and ResNet152, as seen in Table V. PATCHPRO’ running time is reasonable given its superior repair and generalization abilities. Although TRADES performs comparably, PATCHPRO consistently delivers higher performance across various metrics. For both datasets, patch modules are applied at the network’s second-to-last layer, using a fully connected network with a single linear layer. The linear layer takes the output from the network’s penultimate layer as input, and its output is added to the original network’s output.

Answer to RQ3: PATCHPRO demonstrates good scalability in repairing local robustness on large-scale DNNs, which also achieves a 100% repair success rate coupled with 0% drawdown, outperforming the other tools.

E. Impact of Patch Module Size and Quantity on Efficiency

In this experiment, we evaluate the effectiveness of patch module size and quantity in repairing local robustness properties. We perform a comparative analysis of PATCHPRO’s performance across different sizes and quantities, presenting the experimental results in Table VI. The performance of PATCHPRO is assessed using VGG19 and ResNet18 on the CIFAR-10 dataset, across various combinations of patch module scales and quantities, including small patch modules (PS) and large patch modules (PL).

Two distinct patch module sizes are considered: small and large. The small patch module consists of a single linear layer, which takes the output from the network’s penultimate layer as input. The large patch module also takes the output from the penultimate layer as input but additionally comprises three hidden layers with 200 neurons each. It is noted that larger patch

networks tend to yield poorer results, possibly due to increased over approximation error when passing through ReLU nodes in the hidden layers. As the number of hidden layers increases, the cumulative over approximation error grows, leading to less effective repairs. Smaller patch scales slightly reduce the repair time. Additionally, when using DeepPoly to verify the feature layer repairs after applying smaller patches, the verification success rate is higher.

We also tested the effect of using a single patch to repair the neural network, and the results are shown in Table VII. It can be observed that the repair effectiveness using a single patch is significantly inferior to using multiple patches. A single patch only achieves 10% to 20% RSR and RGR. In contrast, using multiple patches results in both RSR and RGR reaching 100%, as shown in Table IV. Additionally, the single patch performs much worse in terms of DSR and DGSR. Therefore, using multiple patches for repair is highly beneficial.

Answer to RQ4: We observe that smaller patch modules are more effective, likely due to reduced over approximation error and shorter repair times, as well as higher verification success rates with abstract interpretation tools. For the aspect of patch module quantity, multiple patch modules significantly outperform only one patch module.

V. RELATED WORK

Provable DNN repair. The methods most closely related to ours are ART [25] and REASSURE [21]. In comparison to REASSURE, our approach fundamentally differs in its repair objectives. REASSURE focuses on fixing activation patterns of neural networks, yet for high-dimensional data, the input constraints of a property may encompass a large number of activation patterns. In contrast, PATCHPRO leverages formal verification to directly repair properties. Although ART also employs training in its repair process, it modifies parameters

TABLE V
RESULTS OF REPAIRING LOCAL ROBUSTNESS ON TINY IMAGENET

Model	r	n	RSR/%			RGR/%			DD/%			DSR/%			DGR/%			Time/s		
			CARE	TRADE	Ours	CARE	TRADE	Ours	CARE	TRADE	Ours	CARE	TRADE	Ours	CARE	TRADE	Ours	CARE	TRADE	Ours
WRN 101-2	$\frac{2}{255}$	500	0.0	100.0	100.0	1.2	100.0	100.0	0.1	46.8	0.0	0.0	100.0	100.0	0.0	6.6	64.4	768.7	5040.8	2029.6
		1000	0.0	100.0	100.0	0.5	100.0	100.0	0.0	43.2	0.0	0.0	100.0	100.0	0.0	7.3	64.4	1345.7	7088.6	3260.7
		500	0.0	100.0	100.0	0.0	100.0	100.0	0.0	51.8	0.0	0.0	97.6	100.0	0.0	1.8	64.4	721.9	4985.5	1731.8
ResNet 152	$\frac{4}{255}$	1000	0.0	100.0	100.0	0.0	100.0	100.0	0.1	49.5	0.0	0.0	97.2	100.0	0.0	2.1	64.4	1440.8	6976.2	3221.2
		500	0.0	100.0	100.0	0.0	100.0	100.0	0.0	64.5	0.0	0.0	100.0	100.0	0.0	5.2	68.2	545.3	4667.7	2219.6
		1000	0.0	100.0	100.0	0.0	100.0	100.0	0.0	62.6	0.0	0.0	99.9	100.0	0.0	6.1	68.2	1092.9	7503.5	4703.7
	$\frac{4}{255}$	500	0.0	100.0	100.0	0.0	100.0	100.0	0.0	64.5	0.0	0.0	97.6	100.0	0.0	1.3	68.2	535.4	4624.2	2336.1
		1000	0.0	100.0	100.0	0.0	100.0	100.0	0.0	63.6	0.0	0.0	98.5	100.0	0.0	2.0	68.2	1028.1	7499.9	4366.8

TABLE VI
THE EFFICACY OF PATCH MODULE SCALE

		VGG19										ResNet-18									
		$r = 4/255$					$r = 8/255$					$r = 4/255$					$r = 8/255$				
		50	100	200	500	1000	50	100	200	500	1000	50	100	200	500	1000	50	100	200	500	1000
DSR/%	PS	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	PL	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	98.0	99.0	99.5	99.2	99.4	100.0	100.0	99.5	99.8	99.8
DGR/%	PS	93.4	93.4	93.4	93.4	93.4	93.4	93.4	93.4	93.4	93.4	88.3	88.3	88.3	88.3	88.3	88.3	88.3	88.3	88.3	88.3
	PL	93.2	93.2	92.8	92.0	92.3	91.2	93.3	93.4	93.4	93.4	60.0	77.5	84.2	81.4	87.2	68.6	74.5	77.3	82.5	86.6
Time/s	PS	238.7	471.6	963.0	2516.9	5441.6	229.9	470.7	950.1	2485.0	5436.0	265.8	525.1	1050.3	2586.4	5262.3	264.7	523.1	1042.8	2627.8	5325.1
	PL	263.3	497.1	987.1	2562.7	4991.7	244.7	495.5	992.9	2468.7	5169.8	292.5	583.4	1155.1	3076.9	5865.6	341.5	592.4	1154	3007.7	5941.1
Verified	PS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	PL	✓	×	×	×	×	×	×	×	×	×	✓	✓	✓	✓	×	×	×	×	×	×

TABLE VII
THE EFFICACY OF SINGLE PATCH MOUDLE

		VGG19										ResNet-18									
		$r = 4/255$					$r = 8/255$					$r = 4/255$					$r = 8/255$				
		50	100	200	500	1000	50	100	200	500	1000	50	100	200	500	1000	50	100	200	500	1000
RSR/%		10.0	18.0	16.5	18.2	16.7	10.0	9.0	14.5	11.8	11.6	12.0	18.0	16.5	16.2	16.2	10.0	8.0	12.0	11.6	11.3
RGR/%		10.0	18.0	16.5	18.2	16.7	10.0	9.0	14.5	11.8	11.6	12.0	18.0	16.5	16.2	16.2	10.0	8.0	12.0	11.6	11.3
DSR/%		6.0	4.0	2.5	1.8	1.3	6.0	6.0	3.0	2.0	1.3	4.0	1.0	3.5	1.8	1.3	8.0	6.0	4.0	4.8	4.1
DGR/%		5.8	5.2	3.9	3.9	3.8	5.7	5.6	2.05	1.9	1.9	5.2	1.5	1.5	1.5	1.4	7.9	7.2	5.4	4.8	4.8

in the original neural network. On the other hand, PATCHPRO uses patch modules specifically to fix properties, ensuring the performance of the original network. Other methods such as PRDNN [20] and APRNN [22] formulate the repair problem in linear programming. However, their efficacy is limited on properties with high-dimensional polytopes, such as the robustness of image classification.

Heuristic DNN repair. Utilizing heuristic algorithms such as particle swarm optimization and differential evolution, CARE [8] and Arachne [9] aim to pinpoint the neurons responsible for faults. VeRe [10] focuses on providing formal verification guidance to assist fault localization, and defines target intervals for repair synthesis. For example, DeepRepair [16] and few-shot guided mix [17] expand the set of negative samples to generate additional training data. Conversely, Tian et al. [41] augment the existing data by introducing real-world environmental effects, such as fog, to the samples. DL2 [24] fuses logical constraints and loss functions, but without convex-certified guarantees.

DNN verification. Over the past decade, various approaches of DNN verification has been proposed including techniques such as constraint solving [42]–[45], abstract interpretation [11], [46]–[49], linear relaxation [50]–[54], global optimisation [55]–[57], CEGAR [58]–[60], reduction to two-player games [61], [62], and star-set abstraction [63], [64]. These method offer provable estimations of DNN robustness. Moreover, statistical approaches, presented in [65]–[74], prove to be more efficient and scalable, particularly suited for intricate DNN structures, allowing for the establishment of quantifiable

robustness at a specified confidence level. Certified training [75]–[77] uses convex approximation in training the loss function, but it is only used to enhance the robustness of DNN, without making it accessible to repair properties in the training procedure.

VI. CONCLUSION

We introduce PATCHPRO, a novel approach for property-based repair of local robustness using limited data. Our method provides patch modules as neural networks to repair within the robustness neighborhood, enabling the generalization of this defense to other inputs. In terms of efficiency, scalability, and generalization, our approach surpasses existing methods.

VII. ACKNOWLEDGEMENTS

This work is supported by CAS Project for Young Scientists in Basic Research, Grant No.YSBR-040, ISCAS Basic Research ISCAS-JCZD-202302, and ISCAS New Cultivation Project ISCAS-PYFX-202201. The limitation of our work is that our patch allocation may assign inappropriate patches when the input is misclassified, and in this case PATCHPRO may fail to repair its local robustness. For future work, we plan to integrate verification methods based on Branch-and-Bound like [36] to enhance the performance of our approach, improving both precision and offering an alternative refinement approach. Additionally, our patch-based repair framework has the potential to evolve into a black-box repair method combined with black-box verification methods like [74].

REFERENCES

- [1] M. Badar, M. Haris, and A. Fatima, "Application of deep learning for retinal image analysis: A review," *Computer Science Review*, vol. 35, p. 100203, 2020.
- [2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.
- [3] Y. Wang, X. Deng, S. Pu, and Z. Huang, "Residual convolutional ctc networks for automatic speech recognition," *arXiv preprint arXiv:1702.07793*, 2017.
- [4] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *ICLR 2018*. Vancouver, BC, Canada: OpenReview.net, 2018.
- [5] F. Croce and M. Hein, "Mind the box: l_1 -apgd for sparse adversarial attacks on image classifiers," in *ICML*, 2021.
- [6] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.
- [7] S. Vieira, W. H. Pinaya, and A. Mechelli, "Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications," *Neuroscience & Biobehavioral Reviews*, vol. 74, pp. 58–75, 2017.
- [8] B. Sun, J. Sun, L. H. Pham, and J. Shi, "Causality-based neural network repair," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 338–349.
- [9] J. Sohn, S. Kang, and S. Yoo, "Arachne: Search based repair of deep neural networks," *ACM Transactions on Software Engineering and Methodology*, 2022.
- [10] J. Ma, P. Yang, J. Wang, Y. Sun, C.-C. Huang, and Z. Wang, "Vere: Verification guided synthesis for repairing deep neural networks," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–13.
- [11] T. Gehr, M. Mirman, D. Drachler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "Ai2: Safety and robustness certification of neural networks with abstract interpretation," in *2018 IEEE symposium on security and privacy (SP)*. IEEE, 2018, pp. 3–18.
- [12] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *The journal of machine learning research*, vol. 17, no. 1, pp. 2096–2030, 2016.
- [13] F. Tramer and D. Boneh, "Adversarial training and robustness for multiple perturbations," *Advances in neural information processing systems*, vol. 32, 2019.
- [14] W. Dai, O. Jin, G.-R. Xue, Q. Yang, and Y. Yu, "Eigentransfer: a unified framework for transfer learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, pp. 193–200.
- [15] W. Ying, Y. Zhang, J. Huang, and Q. Yang, "Transfer learning via learning to transfer," in *International conference on machine learning*. PMLR, 2018, pp. 5085–5094.
- [16] B. Yu, H. Qi, Q. Guo, F. Juefei-Xu, X. Xie, L. Ma, and J. Zhao, "Deeprepair: Style-guided repairing for deep neural networks in the real-world operational environment," *IEEE Transactions on Reliability*, vol. 71, no. 4, pp. 1401–1416, 2021.
- [17] X. Ren, B. Yu, H. Qi, F. Juefei-Xu, Z. Li, W. Xue, L. Ma, and J. Zhao, "Few-shot guided mix for dnn repairing," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2020, pp. 717–721.
- [18] S. Ma, Y. Liu, W.-C. Lee, X. Zhang, and A. Grama, "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*, 2018, pp. 175–186.
- [19] Z. Allen-Zhu and Y. Li, "Feature purification: How adversarial training performs robust deep learning," in *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. Los Alamitos, CA, USA: IEEE Computer Society, feb 2022, pp. 977–988.
- [20] M. Sotoudeh and A. V. Thakur, "Provable repair of deep neural networks," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2021, pp. 588–603.
- [21] F. Fu and W. Li, "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, 2022.
- [22] Z. Tao, S. Nawas, J. Mitchell, and A. V. Thakur, "Architecture-preserving provable repair of deep neural networks," *Proceedings of the ACM on Programming Languages*, vol. 7, no. PLDI, pp. 443–467, 2023.
- [23] G. Singh, T. Gehr, M. Püschel, and M. T. Vechev, "An abstract domain for certifying neural networks," *PACMPL*, vol. 3, no. POPL, pp. 41:1–41:30, 2019.
- [24] M. Fischer, M. Balunovic, D. Drachler-Cohen, T. Gehr, C. Zhang, and M. T. Vechev, "DL2: training and querying neural networks with logic," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019, pp. 1931–1941.
- [25] X. Lin, H. Zhu, R. Samanta, and S. Jagannathan, "Art: Abstraction refinement-guided training for provably correct neural networks," in *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020*. IEEE, 2020, pp. 148–157.
- [26] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," *Advances in neural information processing systems*, vol. 31, 2018.
- [27] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. T. Vechev, "Fast and effective robustness certification," in *NeurIPS 2018*, Montréal, Canada, 2018, pp. 10 825–10 836.
- [28] Z. Chi, J. Ma, P. Yang, C.-C. Huang, R. Li, X. Huang, and L. Zhang, "Patch synthesis for property repair of deep neural networks," 2025. [Online]. Available: <https://arxiv.org/abs/2404.01642>
- [29] J. Wang, G. Dong, J. Sun, X. Wang, and P. Zhang, "Adversarial sample detection for deep neural network through model mutation testing," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1245–1256.
- [30] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. [Online]. Available: <https://doi.org/10.1109/5.726791>
- [31] A. Krizhevsky, "Learning multiple layers of features from tiny images," Jan 2009.
- [32] Y. Le and X. S. Yang, "Tiny imagenet visual recognition challenge," 2015. [Online]. Available: <https://api.semanticscholar.org/CorpusID:16664790>
- [33] C. von Essen and D. Giannakopoulou, "Analyzing the next generation airborne collision avoidance system," in *TACAS 2014*, ser. Lecture Notes in Computer Science, E. Ábrahám and K. Havelund, Eds., vol. 8413. Springer, 2014, pp. 620–635.
- [34] J. Jeannin, K. Ghorbal, Y. Kouskoulas, R. Gardner, A. Schmidt, E. Zawadzki, and A. Platzer, "Formal verification of ACAS x, an industrial airborne collision avoidance system," in *EMSOFT 2015*, A. Girault and N. Guan, Eds. IEEE, 2015, pp. 127–136.
- [35] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *ICLR 2015*, 2015.
- [36] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *CAV 2017*. Heidelberg, Germany: Springer, 2017, pp. 97–117.
- [37] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: <http://arxiv.org/abs/1409.1556>
- [38] R. Ferjaoui, M. A. Cherni, F. Abidi, and A. Zidi, "Deep residual learning based on resnet50 for COVID-19 recognition in lung CT images," in *8th International Conference on Control, Decision and Information Technologies, CoDIT 2022, Istanbul, Turkey, May 17-20, 2022*. IEEE, 2022, pp. 407–412. [Online]. Available: <https://doi.org/10.1109/CoDIT55151.2022.9804094>
- [39] S. Zagoruyko and N. Komodakis, "Wide residual networks," *CoRR*, vol. abs/1605.07146, 2016. [Online]. Available: <http://arxiv.org/abs/1605.07146>
- [40] H. Zhang, Y. Yu, J. Jiao, E. Xing, L. El Ghaoui, and M. Jordan, "Theoretically principled trade-off between robustness and accuracy," in *International conference on machine learning*. PMLR, 2019, pp. 7472–7482.

- [41] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*, 2018, pp. 303–314.
- [42] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24–28, 2017, Proceedings, Part I* 30. Springer, 2017, pp. 97–117.
- [43] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic, D. L. Dill, M. J. Kochenderfer, and C. W. Barrett, "The marabou framework for verification and analysis of deep neural networks," in *CAV 2019*, ser. Lecture Notes in Computer Science, I. Dillig and S. Tasiran, Eds., vol. 11561. New York City, NY, USA: Springer, 2019, pp. 443–452.
- [44] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *ATVA 2017*. Pune, India: Springer, 2017, pp. 269–286.
- [45] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *CAV 2017*. Heidelberg, Germany: Springer, 2017, pp. 3–29.
- [46] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–30, 2019.
- [47] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and effective robustness certification," *Advances in neural information processing systems*, vol. 31, 2018.
- [48] J. Li, J. Liu, P. Yang, L. Chen, X. Huang, and L. Zhang, "Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification," in *Static Analysis: 26th International Symposium, SAS 2019, Porto, Portugal, October 8–11, 2019, Proceedings* 26. Springer, 2019, pp. 296–319.
- [49] G. Singh, R. Ganvir, M. Püschel, and M. T. Vechev, "Beyond the single neuron convex barrier for neural network certification," in *NeurIPS 2019*, 2019, pp. 15 072–15 083.
- [50] K. D. Julian, S. Sharma, J.-B. Jeannin, and M. J. Kochenderfer, "Verifying aircraft collision avoidance neural networks through linear approximations of safe regions," *arXiv preprint arXiv:1903.00762*, 2019.
- [51] B. Paulsen, J. Wang, and C. Wang, "Reludiff: Differential verification of deep neural networks," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 714–726.
- [52] K. Xu, Z. Shi, H. Zhang, Y. Wang, K.-W. Chang, M. Huang, B. Kailkhura, X. Lin, and C.-J. Hsieh, "Automatic perturbation analysis for scalable certified robustness and beyond," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1129–1141, 2020.
- [53] T. Weng, H. Zhang, H. Chen, Z. Song, C. Hsieh, L. Daniel, D. S. Boning, and I. S. Dhillon, "Towards fast computation of certified robustness for relu networks," in *ICML 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. Stockholm, Sweden: PMLR, 2018, pp. 5273–5282.
- [54] B. Batten, P. Kouvaros, A. Lomuscio, and Y. Zheng, "Efficient neural network verification via layer-based semidefinite relaxations and linear cuts," *IJCAI*, 2021.
- [55] W. Ruan, X. Huang, and M. Kwiatkowska, "Reachability analysis of deep neural networks with provable guarantees," in *IJCAI 2018*. Stockholm, Sweden: ijcai.org, 2018, pp. 2651–2659.
- [56] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Output range analysis for deep feedforward neural networks," in *NFM 2018*, ser. Lecture Notes in Computer Science, A. Dutle, C. A. Muñoz, and A. Narkawicz, Eds., vol. 10811. Newport News, VA, USA: Springer, 2018, pp. 121–138.
- [57] W. Ruan, M. Wu, Y. Sun, X. Huang, D. Kroening, and M. Kwiatkowska, "Global robustness evaluation of deep neural networks with provable guarantees for the hamming distance," in *IJCAI 2019*, S. Kraus, Ed. Macao, China: ijcai.org, 2019, pp. 5944–5952.
- [58] P. Ashok, V. Hashemi, J. Kretínský, and S. Mohr, "Deepabstract: Neural network abstraction for accelerating verification," in *ATVA 2020*, ser. Lecture Notes in Computer Science, vol. 12302. Springer, 2020, pp. 92–107.
- [59] Y. Y. Elboher, J. Gottschlich, and G. Katz, "An abstraction-based framework for neural network verification," in *CAV 2020*, ser. Lecture Notes in Computer Science, S. K. Lahiri and C. Wang, Eds., vol. 12224. Los Angeles, CA, USA: Springer, 2020, pp. 43–65.
- [60] M. Ostrovsky, C. W. Barrett, and G. Katz, "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*, ser. Lecture Notes in Computer Science, vol. 13505. Springer, 2022, pp. 391–396.
- [61] M. Wicker, X. Huang, and M. Kwiatkowska, "Feature-guided black-box safety testing of deep neural networks," in *TACAS 2018*, ser. Lecture Notes in Computer Science, D. Beyer and M. Huisman, Eds., vol. 10805. Thessaloniki, Greece: Springer, 2018, pp. 408–426.
- [62] M. Wu, M. Wicker, W. Ruan, X. Huang, and M. Kwiatkowska, "A game-based approximate verification of deep neural networks with provable guarantees," *Theor. Comput. Sci.*, vol. 807, pp. 298–329, 2020.
- [63] H. Tran, D. M. Lopez, P. Musau, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson, "Star-based reachability analysis of deep neural networks," in *FM 2019*, ser. Lecture Notes in Computer Science, M. H. ter Beek, A. McIver, and J. N. Oliveira, Eds., vol. 11800. Porto, Portugal: Springer, 2019, pp. 670–686.
- [64] H. Tran, S. Bak, W. Xiang, and T. T. Johnson, "Verification of deep convolutional neural networks using imagestars," in *CAV 2020*, ser. Lecture Notes in Computer Science, S. K. Lahiri and C. Wang, Eds., vol. 12224. Los Angeles, CA, USA: Springer, 2020, pp. 18–42.
- [65] S. Webb, T. Rainforth, Y. W. Teh, and M. P. Kumar, "A statistical approach to assessing neural network robustness," in *ICLR 2019*. New Orleans, LA, USA: OpenReview.net, 2019.
- [66] L. Weng, P. Chen, L. M. Nguyen, M. S. Squillante, A. Boopathy, I. V. Oseledets, and L. Daniel, "PROVEN: verifying robustness of neural networks with a probabilistic approach," in *ICML 2019, 9–15 June 2019*, ser. Proceedings of Machine Learning Research, vol. 97. Long Beach, California, USA: PMLR, 2019, pp. 6727–6736.
- [67] T. Baluta, Z. L. Chua, K. S. Meel, and P. Saxena, "Scalable quantitative verification for deep neural networks," in *ICSE 2021*. Madrid, Spain: IEEE, 2021, pp. 312–323.
- [68] M. Wicker, L. Laurenti, A. Patane, and M. Kwiatkowska, "Probabilistic safety for bayesian neural networks," in *UAI 2020, August 3–6, 2020*, ser. Proceedings of Machine Learning Research, R. P. Adams and V. Gogate, Eds., vol. 124. virtual online: AUAI Press, 2020, pp. 1198–1207.
- [69] T. Baluta, S. Shen, S. Shinde, K. S. Meel, and P. Saxena, "Quantitative verification of neural networks and its security applications," in *CCS 2019, November 11–15, 2019*. London, UK: ACM, 2019, pp. 1249–1264.
- [70] L. Cardelli, M. Kwiatkowska, L. Laurenti, N. Paoletti, A. Patane, and M. Wicker, "Statistical guarantees for the robustness of bayesian neural networks," in *IJCAI 2019, August 10–16, 2019*, S. Kraus, Ed. Macao, China: ijcai.org, 2019, pp. 5693–5700.
- [71] P. Huang, Y. Yang, M. Liu, F. Jia, F. Ma, and J. Zhang, "e-weakened robustness of deep neural networks," *CoRR*, vol. abs/2110.15764, 2021.
- [72] R. Mangal, A. V. Nori, and A. Orso, "Robustness of neural networks: a probabilistic and practical approach," in *ICSE (NIER) 2019, Montreal, QC, Canada, May 29–31, 2019*. Montreal, QC, Canada: IEEE / ACM, 2019, pp. 93–96.
- [73] L. Cardelli, M. Kwiatkowska, L. Laurenti, and A. Patane, "Robustness guarantees for bayesian inference with gaussian processes," in *AAAI 2019, January 27 - February 1, 2019*. Honolulu, Hawaii, USA: AAAI Press, 2019, pp. 7759–7768.
- [74] R. Li, P. Yang, C. Huang, Y. Sun, B. Xue, and L. Zhang, "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, 2022, pp. 2189–2201. [Online]. Available: <https://doi.org/10.1145/3510003.3510143>
- [75] M. Balunović and M. Vechev, "Adversarial training and provable defenses: Bridging the gap," in *8th International Conference on Learning Representations (ICLR 2020)(virtual)*. International Conference on Learning Representations, 2020.
- [76] Y. Mao, M. Müller, M. Fischer, and M. Vechev, "Connecting certified and adversarial training," *Advances in Neural Information Processing Systems*, vol. 36, 2024.
- [77] M. N. Müller, F. Eckert, M. Fischer, and M. T. Vechev, "Certified training: Small boxes are all you need," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1–5, 2023*. OpenReview.net, 2023.