Provable Repair of Deep Neural Network Defects by Preimage Synthesis and Property Refinement

Jianan Ma*
Hangzhou Dianzi University
Hangzhou, China
Zhejiang University
Hangzhou, China
majianannn@gmail.com

Jingyi Wang†
Zhejiang University
Hangzhou, China
wangjyee@zju.edu.cn

Qi Xuan
Institute of Cyberspace Security,
Zhejiang University of Technology
Hangzhou, China
Binjiang Institute of AI, ZJUT
Hangzhou, China
xuanqi@zjut.edu.cn

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

Abstract

It is known that deep neural networks may exhibit dangerous behaviors under various security threats (e.g., backdoor attacks, adversarial attacks and safety property violation) and there exists an ongoing arms race between attackers and defenders. In this work, we propose a complementary perspective to utilize recent progress on "neural network repair" to mitigate these security threats and repair various kinds of neural network defects (arising from different security threats) within a unified framework, offering a potential silver bullet solution to real-world scenarios. To substantially push the boundary of existing repair techniques (suffering from limitations such as lack of guarantees, limited scalability, considerable overhead, etc) in addressing more practical contexts, we propose ProRepair, a novel *provable* neural network repair framework driven by formal preimage synthesis and property refinement. The key intuitions are: (i) synthesizing a precise proxy box to characterize the feature space preimage, which can derive a bounded distance term sufficient to guide the subsequent repair step towards the correct outputs, and (ii) performing property refinement to enable surgical corrections and scale to more complex tasks. We evaluate ProRepair across four security threats repair tasks on six benchmarks and the results demonstrate it outperforms existing methods in effectiveness, efficiency and scalability. For point-wise repair, ProRepair corrects models while preserving performance and achieving significantly improved generalization, with a speed-up of 5× to 2000× over existing provable approaches. In region-wise repair, ProRepair successfully repairs all 36 safety property violation instances (compared to 8 by the best existing method), and can handle 18× higher dimensional spaces.

CCS Concepts

• **Computing methodologies** → **Machine learning**; • **Security and privacy** → **Software and application security**.

Keywords

AI security; Neural network repair; Preimage synthesis

1 Introduction

Numerous studies have demonstrated that deep neural networks (DNNs) are inherently vulnerable to a wide range of security threats, including adversarial attacks [16, 53], natural corruption [21, 56], backdoor attacks [1, 5, 33], and violations of safety properties [25, 27], all of which may lead to dangerous and unpredictable results in safety-critical applications. The evolving nature of security threats has triggered an ongoing arms race between attackers and defenders. Attackers continuously enhance their strategies to bypass existing defenses, while defenders respond by developing new techniques to address emerging risks, such as adversarial training [14, 30, 51] and defense [9, 59, 66, 70], backdoor detection [38, 54, 63] and removal [15, 55, 71]. These defenses are typically tailored to specific threats and applied before model deployment, making them inadequate to address unforeseen failures that arise after the model is deployed. As a consequence, DNNs deployed in real-world environments remain vulnerable to unexpected failures. This raises a crucial question: *how to utilize the failures arising from diverse security threats after the model is deployed to further enhance its safety and reliability?*

Recent advances in **neural network (NN) repair** offer a promising silver bullet solution to this critical problem. Unlike existing defenses, NN repair aims to handle post-deployment defects that may arise from various security threats through iterative error-driven updates. This enables a unified and adaptive framework that is particularly suitable for real-world scenarios. Building on its potential, a number of NN repair methods have been proposed, progressively improving effectiveness, efficiency and applicability across diverse security threats. Formally, given a property ϕ that defines the desired behavior as a tuple of input set ϕ^{in} and output space ϕ^{out}, NN repair aims to correct a buggy model f to

*Part of Jianan Ma's work was done when visiting Zhejiang University.
†Corresponding author: Jingyi Wang.

This is the full version of the paper accepted by CCS 2025.

$\tilde{f}$ such that $\forall \boldsymbol{x} \in \phi^{in},\ \tilde{f}(\boldsymbol{x}) \in \phi^{out}$. For example, the property ϕ may encode specifications like robustness to bounded perturbations, where $\phi^{in} = \{\boldsymbol{x} \mid \|\boldsymbol{x} - \boldsymbol{x}_0\| \leq \delta\}$ defines a local space and $\phi^{out} = \{\boldsymbol{y} \mid \boldsymbol{y}_{true} \geq \boldsymbol{y}_{other}\}$ ensures correct classification.

Existing solutions to this problem fall into two categories: heuristic repair based on fault localization and provable repair with theoretical guarantees. The first category focuses on identifying the key units in the model that are responsible for errors, which typically leverage gradients [52], activation values [46], or causal models [49] to pinpoint faulty neurons or parameters. However, they lack theoretical guarantees (the repaired NN may still violate ϕ) and cannot handle region-wise properties (where ϕ^{in} is an infinite set). To address these issues, recent *provable repair* techniques [13, 47, 50] leverage the piecewise-linear nature of ReLU NNs: if the activation status (active or inactive for ReLU) of all neurons are fixed, the entire model can be expressed as a linear function of the inputs or parameters, allowing the repair to be converted into a linear programming (LP), which can be solved by constraint solvers.

Research gap – While prior provable works have been proven effective in certain cases, they still face several key limitations when applied to more practical contexts: 1) *Limited scalability.* Existing methods enforce fixed activation status throughout the input space ϕ^{in}. This rigid requirement severely limits their capability to find feasible solutions, especially when there are multiple properties to repair or for high-dimensional spaces. Additionally, the dependence on activation status restricts their applicability to NNs with piecewise-linear activation functions. 2) *High computational cost.* To ensure the model behaves linearly in ϕ^{in}, they must enumerate either all activation status or all vertices of the space, which results in exponential complexity with respect to its dimensionality. This produces extremely large LP requiring massive computational resources. For example, [50] (SOTA) builds an LP with over a billion coefficients when repairing in a 16D space, which takes over 10,000 seconds and 200 GB of memory to solve. 3) *Unstable Fidelity preservation.* Since modifying earlier layers would cause uncontrolled activation status in all subsequent layers, they are limited to repairing only the final few layers. This restriction compels the LP solution to make excessive modifications on these layers, which may significantly degrade the model's original performance.

Insight - Instead of modifying the final few layers (for decision making), we propose an inverse perspective that focuses on repairing the early layers (for feature extraction). The intuition is that overwhelming modifications to later layers cause dramatic shifts on the feature-space preimage (the set of features that produce desired outputs when fed to subsequent layers), thereby compromising the model performance. This motivates us to characterize the feature space preimage, which can guide the correction of the early layers while leaving the preimage itself unchanged. To further address the repair cost and scalability issues, we propose an adaptive space refinement approach which employs NN verification techniques to eliminate activation status dependency while providing formal repair guarantees. By adaptively refining the input space, the approximation errors induced by verification are effectively reduced, with each subspace receiving more surgical corrections that collectively enable the repair of more complex tasks.

Solution - Based on the above insights, we design and implement ProRepair, a novel provable NN repair framework to mitigate diverse kinds of security threats. As shown in Fig. 1, ProRepair first generates counterexamples for the given desired properties, which are then used to synthesize preimages. Since the numerous nonlinear units in the NN make exact synthesis impractical, ProRepair proposes to employ linear relaxation with iterative center shift to generate a small proxy box to precisely and efficiently characterize the preimage. The geometric nature of proxy boxes further enables us to derive a bounded distance measure carefully crafted for repair. After one repair iteration, ProRepair selects a critical dimension to split for each space that fails verification, allowing space refinement for surgical repair.

Table 1: A comparison of different NN repair methods. ■ and □ denote the method supports the attribute or not. △: REASSURE only supports ReLU. ▲: PRDNN theoretically supports any activation function for point-wise repair and piece-wise linear functions for region-wise repair, but the implementation only supports ReLU. ♢: APRNN supports activation functions that have linear pieces such as ReLU. -, +, and ++ denotes low, medium, and high efficiency, respectively.

Method	Provable	No Extra Data	Architecture Preserving	Region Repair	Complex Activation Functions	Efficiency
NNREPAIR [52]	□	□	■	□	□	-
Arachne [46]	□	□	■	□	■	+
CARE [49]	□	□	■	□	■	+
I-REPAIR [22]	□	□	■	□	■	++
VeRe [34]	□	□	■	□	■	+
AI-Lancet [68]	□	■	■	□	■	+
REASSURE [13]	■	■	□	△	□	-
PRDNN [47]	■	■	□	▲	▲	-
APRNN [50]	■	■	■	♢	♢	-
This work	■	■	■	■	■	++

We have implemented ProRepair as a self-contained toolkit and evaluated it across four security threats repair tasks using various benchmarks and models. The results demonstrate its effectiveness and efficiency, with significant improvements in region-wise repair capability. For point-wise repair, ProRepair preserves model fidelity while achieving remarkable generalization, showcasing a speed improvement of 5× to 2000× over existing provable methods. For region-wise repair, it successfully repairs all 36 safety property violation instances (compared to only 8 repairs by the state-of-the-art method) and can handle 18× higher-dimensional spaces. A comprehensive comparison with existing approaches is presented in Table 1. In summary, our main contributions are:

- We devise a new repair perspective that focuses on the feature extractor, enabled by a novel proxy box synthesis method to capture a meaningful subset of the preimage.
- We propose ProRepair, a novel provable NN repair framework that integrates a bounded distance measure with adaptive property refinement to correct NNs with provable guarantees in more practical contexts.
- Extensive experiments on 4 repair tasks and 76 models across 6 benchmark datasets demonstrate that our approach significantly outperforms existing provable repair methods in terms of effectiveness, efficiency and scalability.

- We release ProRepair at this **repository** along with all the codes, datasets and models to facilitate future studies in this area.

2 Preliminaries

2.1 Deep neural networks

A DNN can be defined as a function $f : \mathbb{R}^m \to \mathbb{R}^n$ maps a high-dimensional input $\boldsymbol{x} \in \mathbb{R}^m$ to an output $f(\boldsymbol{x}) \in \mathbb{R}^n$, where n is the total number of classes. A typical DNN includes multiple hidden layers f_e to extract feature representations, followed by fewer layers f_c to classify based on the extracted features, represented as $f = f_c \circ f_e$. Finally, it chooses the dimension with the highest score as the classification result, i.e., $\arg\max_{1\le i\le n} f(\boldsymbol{x})_i$. We further define the specification of the desirable property that DNNs should satisfy.

Definition 2.1 (**Desired Property Specification**). *For a neural network $f : \mathbb{R}^m \to \mathbb{R}^n$, a desired property ϕ is a triple $(\phi^{in}, \phi^{cons}, \phi^{out})$, where $\phi^{in} \subseteq \mathbb{R}^m$ defines the input space of interest. The output specification is governed by the constraint set $\phi^{cons} = \{\psi^1, \ldots, \psi^q\}$ where each linear constraint ψ enforces $\boldsymbol{c}_\psi^\top f(\cdot) + d_\psi \ge 0$ with $\boldsymbol{c}_\psi \in \mathbb{R}^n$ and $d_\psi \in \mathbb{R}$. This induces the valid output space $\phi^{out} = \{\boldsymbol{y} \in \mathbb{R}^n \mid \bigwedge_{\psi\in\phi^{cons}} (\boldsymbol{c}_\psi^\top \boldsymbol{y} + d_\psi \ge 0)\}$.*

In this work, we consider ϕ^{in} to be a bounded box $\{\boldsymbol{x} \in \mathbb{R}^m \mid \forall i \in [1, m] : \boldsymbol{l}_i^\phi \le \boldsymbol{x}_i \le \boldsymbol{u}_i^\phi\}$ defined by lower and upper bounds $\boldsymbol{l}^\phi, \boldsymbol{u}^\phi \in \mathbb{R}^m$, which is commonly used in NN repair [13, 47, 50]. Note that this includes the special case of a single input point when $\boldsymbol{l}^\phi = \boldsymbol{u}^\phi$. We write $f \models \phi$ when $f(\boldsymbol{x}) \in \phi^{out}$ holds for all $\boldsymbol{x} \in \phi^{in}$, and extend this notation to the property set $\mathcal{P}$ with $f \models \mathcal{P}$ meaning $\forall \phi \in \mathcal{P} : f \models \phi$. A repair is classified as *region-wise* if $\exists \phi \in \mathcal{P}$ where ϕ^{in} contains infinitely many points, otherwise *point-wise*.

2.2 Neural network verification

Neural network verification aims to rigorously prove or disprove whether a given network satisfies certain properties, such as correctness, robustness and fairness. Recently, numerous verification frameworks have been proposed, which can be broadly categorized into *complete* and *incomplete* verification. Complete verification typically based on constraint solving [11, 23, 27, 40], which can give a definite 'yes/no' result in sufficient time. However, verifying NNs with both soundness and completeness is a NP-hard problem [25], making it challenging to terminate in a reasonable time. As a result, such solver-based verification can only scale to DNNs with hundreds of neurons. In practice, incomplete verification strikes a balance between precision and efficiency, typically based on linear relaxation [2, 24, 58, 61, 62] and abstract interpretation [14, 44, 45, 64]. This category of approaches can scale up to medium size models, which contain around 10^5 neurons.

Here we use auto-LiRPA [61] (one of the most widely used verifiers) as an example to briefly illustrate how linear relaxation works. The key idea is to over-approximate the outputs of non-linear activation units with two linear functions as lower/upper bound. For example, an unstable ReLU neuron $z' = \mathrm{ReLU}(z)$ with its input range $[l, u]$ $(l < 0 < u)$ can be relaxed as: $k \cdot z \le z' \le \frac{u(z-l)}{u-l}$, $k \in [0, 1]$. Then the numerical lower bound of z' can be calculated by substituting z in the term $k \cdot z$ with its affine bounds. This process is iteratively applied, with each newly introduced variable is replaced by its affine bounds, until all variables within the expression are reduced to input variables. In this way, the output of f over a given space $\mathcal{B} \subset \mathbb{R}^m$ can be linearly relaxed as: $\forall \boldsymbol{x} \in \mathcal{B}$, $\underline{\boldsymbol{w}} \cdot \boldsymbol{x} + \underline{\boldsymbol{b}} \le f(\boldsymbol{x}) \le \overline{\boldsymbol{w}} \cdot \boldsymbol{x} + \overline{\boldsymbol{b}}$. This recursive process of propagating affine bounds layer by layer is commonly referred to as *bound propagation*, and forms the basis of many incomplete verifiers such as auto-LiRPA and Deeppoly [45].

2.3 Preimage

Given an input space $\mathcal{I}$ and an output space $\mathcal{O}$, the preimage is the set of all inputs $\boldsymbol{x} \in \mathcal{I}$ that are mapped to an element in $\mathcal{O}$ by a function f. Formally, we define the preimage for NNs as:

Definition 2.2 (**Neural network preimage**). *Given a DNN $f : \mathbb{R}^m \to \mathbb{R}^n$, an input space $\mathcal{I} \subset \mathbb{R}^m$ and an output space $\mathcal{O} \subset \mathbb{R}^n$, the preimage is defined as: $f^{-1}(\mathcal{I}, \mathcal{O}) = \{\boldsymbol{x} \in \mathcal{I} \mid f(\boldsymbol{x}) \in \mathcal{O}\} \subset \mathbb{R}^m$.*

Generating the exact preimage for DNNs [37] is intractable due to the presence of numerous nonlinear units. Recently, some works have focused on under- and over-approximating the preimage. For instance, [28] utilizes lagrangian dual optimization for preimage over-approximations while [67] presents an anytime method to produce under-approximation of preimage. Overall, they yield a convex approximation of the preimage, i.e., $\mathcal{U} \subseteq f^{-1}(\mathcal{I}, \mathcal{O}) \subseteq \mathcal{T}$, where $\mathcal{U}$ and $\mathcal{T}$ are convex sets.

3 Threat Model

We consider a post-deployment scenario where the defender has deployed a model which exhibits failures in operation, potentially triggered by the following security threats (illustrated in Fig. 1, left).
Threat 1: Adversarial Perturbation. For predetermined inputs (e.g., specific traffic signs), adversaries can craft imperceptible perturbations under ℓ_∞-norm bounds to force misclassification, which are particularly hazardous when targeting safety-sensitive inputs (e.g., perturbing stop signs to be misclassified as yield signs) [12].
Threat 2: Natural Corruption. Deployed models frequently encounter reliability degradation due to real-world environmental disturbances that deviate from training conditions [18, 60]. These disturbances manifest as input corruptions like weather effects (fog, rain) and sensor noise (motion blur, brightness variations), revealing the fragility of DNNs trained on idealized datasets [20].
Threat 3: Backdoor Attack. Due to the growing popularity of online machine learning platforms [38], backdoor attacks have become a significant concern. In line with the classic threat model from existing backdoor learning studies [15, 63, 71], we assume the defender deploys a third-party model implanted with malicious triggers of arbitrary shape/size.
Threat 4: Safety Property Violation. In this case, we consider security-critical systems that rely on DNNs for decision making, such as ACAS Xu [36], an airborne collision avoidance system for unmanned aircraft. In these systems, violations of established safety properties [7, 27], like failure to prevent a collision, pose significant security risks. We follow the general threat model from the neural network verification domain, assuming the defender deploys a model that may potentially violate safety properties.
Defender's Capabilities. We assume the defender has white-box access to the target model (architecture and parameters) and can derive desired properties from multiple sources. For runtime-observed

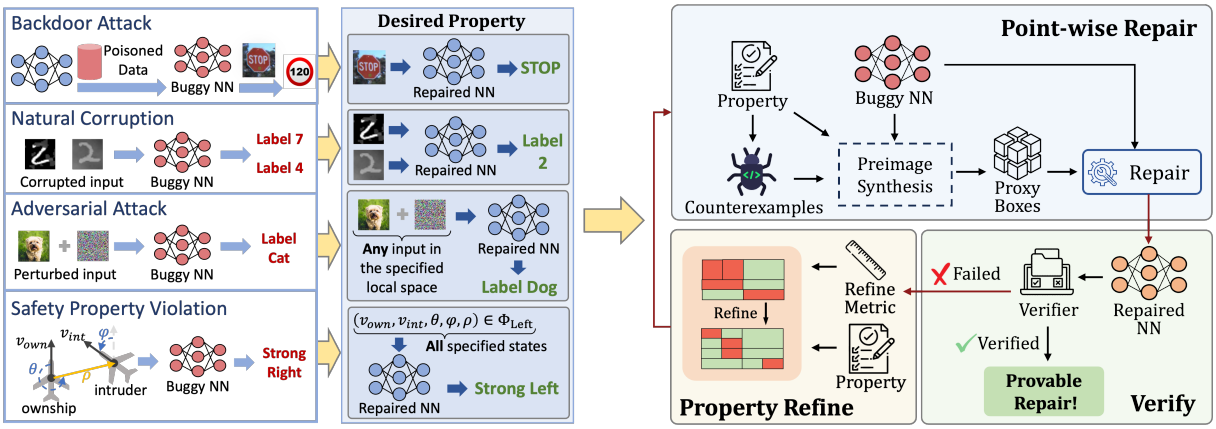


Figure 1: The overview of ProRepair framework.

failures (e.g., a backdoor-triggered stop sign misclassification), the defender can extract the concrete counterexample $\phi^{in} = \{x_0\}$ and formulate constraints $\phi^{cons} = \{\psi^j \mid \psi^j := f(x)_{y_0} - f(x)_j \geq 0, \forall j\}$, where y_0 is the class "STOP". Another common source is developer-specified requirements in safety-critical systems, where the defender can encode domain knowledge as formal properties. With ACAS Xu as a canonical example, the defender may require the model to output "Clear-of-Conflict" (COC) decisions when the intruder's relative distance exceeds safety threshold d_{min} and its velocity is below danger threshold v_{max}, formally expressed as $\phi^{in} = \{x \mid x_{distance} \geq d_{min} \wedge x_{velocity} \leq v_{max}\}$ and $\phi^{cons} = \{\psi^j \mid \psi^j := f(x)_{\text{COC}} - f(x)_j \geq 0, \forall j\}$. These formulations accommodate both reactive repairs and proactive safety enforcement without requiring additional training data, aligning with established repair frameworks [13, 47, 50].

Goals. Given a model f and a set of desired properties $\mathcal{P}$ where $f \not\models \mathcal{P}$, the defender aims to repair it to $\tilde{f}$ such that $\tilde{f} \models \mathcal{P}$. We also aim for the repair to be as efficient as possible while preserving the model's original performance.

4 Related Work

Recent NN repair methods can be categorized into non-provable and provable repair depending on whether they guarantee that the repaired network satisfies the desired properties. Tab. 1 summarizes the comparison of our method and existing repair frameworks.

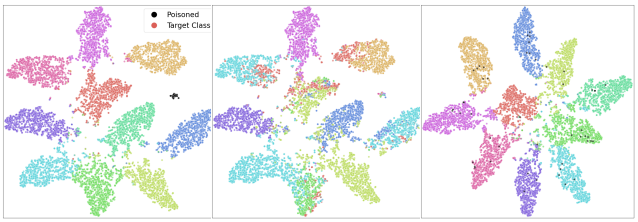
Non-provable repair. Most of these methods start with fault localization, aiming to identify neurons [4, 49] or parameters [6, 22, 46, 52] that are strongly correlated with the bugs. This is typically followed by a repair step, where heuristic algorithms modify the identified neurons or parameters. As they are generally statistics-based, their effectiveness heavily depends on the availability of large amounts of data. More recently, [34] introduced a verification-guided synthesis framework, which improves localization precision when the available data is limited. However, it still requires additional clean data for effective repair. Although these methods generally demonstrate high efficiency and scalability, they are unable to provide theoretical guarantees and handle region-wise property.

Provable repair. Several approaches use formal methods such as constraint solving to ensure that the repaired NN rigorously satisfies the desired properties, providing guarantees for repair. Notably, PRDNN [47], REASSURE [13] and APRNN [50] are recently proposed representative methods. The core challenge faced by these methods is how to transform the repair to linear programming, since directly modifying the parameters in hidden layers introduces nonlinear effects on the model's outputs that existing solvers struggle to handle. Their key idea involves first stabilizing the model's activation status on the given input space so that the model's output can be formalized as a linear transformation of $\Delta\theta$, where $\Delta\theta$ denotes parameter modifications in few layers of f_c (PRDNN and APRNN) or the parameters of the patch network (REASSURE). Notably, they adjust the parameters of f_c because this does not affect the neuron activation status in f_e, while modifying f_e would simultaneously alter the activation status of neurons in f_c. Finally, they invoke a solver to calculate $\Delta\theta$ to correct the model.

However, existing provable methods inherently depend on activation status, which limits applicability to NNs with piecewise linear activation functions and causes scalability issues in more challenging tasks. Specifically, the rigid requirement of identical activation status across the entire input space not only severely limits their capability to find feasible solutions, but also forces vertex enumeration for region-wise repair, which leads to prohibitive computational costs. Consequently, even state-of-the-art methods like APRNN require over 10,000 seconds and 200 GB of memory to repair a single 16D input region. Moreover, focusing on modifying the parameters of f_c to satisfy all the constraints in the constructed LP often significantly compromises model's performance. We conducted a backdoor repair case study to explain this phenomenon based on preimage analysis. As shown in Fig. 2, while APRNN keeps the extracted features unchanged, it drastically distorts the feature space preimages, resulting in nearly 40% accuracy degradation.

5 Methodology

Motivated by the preimage analysis in Sec. 4, we devise a new perspective on repair: instead of modifying the parameters of f_c (which



(a) Original (b) APRNN (c) Ours

Figure 2: t-SNE visualization of feature representations where each group of colored points can be approximately represented as the feature space preimage (set of all features mapping to correct outputs) for the corresponding class. (a) Original network shows well-separated clusters indicating high accuracy; (b) APRNN-repaired net exhibits merged clusters (40% accuracy drop) despite identical feature extraction; (c) Our method repairs the feature extractor while keeping the well-separated preimages unchanged (1.5% accuracy drop).

changes the preimages in the feature space), we focus on correcting f_e, to ensure that the extracted features directly fall into the preimage of the corresponding property. Fig. 2c shows an example (we continue to mark the features of poisoned inputs in black for clarity). We can see that after repair, the features extracted by f_e for the given backdoored inputs are all located within the correct preimages (clusters). Note that the positions of the clusters change due to t-SNE adjusting the layout based on the input features, but the preimages in the feature space remain unchanged.

5.1 Point-wise Repair

In point-wise repair, the input space ϕ^{in} of each property $\phi \in \mathcal{P}$ consists of a single point $\boldsymbol{x}_\phi$. As described in Sec. 4, we expect that the feature of $\boldsymbol{x}_\phi$ extracted by the repaired model to be positioned within the feature space preimage, which can be accomplished in two steps: (1) *calculate the preimage of property* ϕ and f_c, and (2) *modify the feature extractor* f_e *to ensure* $f_e(\boldsymbol{x}_\phi)$ *falls within the preimage.* According to the definition 2.2, we can directly define the exact feature space preimage based on ϕ^{out} and f_c as:

$$f_c^{-1}(\mathbb{R}^s, \phi^{out}) = \left\{\boldsymbol{h} \in \mathbb{R}^s \mid f_c(\boldsymbol{h}) \in \phi^{out}\right\} \tag{1}$$

where s denotes the dimensionality of the feature space. Since exactly calculating the preimage is intractable, a natural solution here for step (1) is to utilize existing tools like [28, 67] to generate the under-approximation $\mathcal{U}$ of the preimage $f_c^{-1}(\mathbb{R}^s, \phi^{out})$, which can further provide valuable information for step (2). For instance, such under-approximate preimage can be leveraged to optimize f_e to minimize the distance between $\mathcal{U}$ and $f_e(\boldsymbol{x}_\phi)$, i.e.,

$$\min_{f_e} \left\| f_e(\boldsymbol{x}_\phi) - \Pi_{\mathcal{U}} f_e(\boldsymbol{x}_\phi) \right\| \tag{2}$$

where Π is the projection operator onto $\mathcal{U}$.

However, existing preimage approximation frameworks typically require the input space to be a bounded box. In our scenario, the input space is the entire unbounded feature space $\mathbb{R}^s$, which makes it impractical to directly apply these frameworks. In addition, the under-approximation $\mathcal{U}$ obtained from these frameworks is

Algorithm 1: Point-wise Repair

Input: DNN $f = f_c \circ f_e$, property set $\mathcal{P}$, box radius r, max iterations for box synthesis T_b, max iterations for repair T_r

Output: repaired model $\tilde{f}$ that $\tilde{f} \models \mathcal{P}$ or failure $\perp$

1 **Function** `Proxy Box Synthesis`$(f_c, \boldsymbol{h}_\phi, \phi, r, T_b)$**:**
2 $\quad \boldsymbol{\alpha}, lb \leftarrow \textsc{LinearBounds}(f_c, \mathcal{B}(\boldsymbol{h}_\phi, r), \phi^{cons})$
$\quad \triangleright\ \boldsymbol{\alpha} : \{\boldsymbol{\alpha}_\psi \in \mathbb{R}^s \mid \psi \in \phi^{cons}\}, lb : \{lb_\psi \in \mathbb{R} \mid \psi \in \phi^{cons}\}$
3 $\quad$ **for** 1 **to** T_b **do**
4 $\quad\quad$ **if** $\min_{\psi \in \phi^{cons}} lb_\psi \geq 0$ **then return** $\boldsymbol{h}_\phi$
5 $\quad\quad \boldsymbol{h}_\phi \leftarrow \arg\max_{\boldsymbol{h} \in \mathcal{B}(\boldsymbol{h}_\phi, r)} \sum_{\psi \in \phi^{cons}} \left(\mathcal{I}_{lb_\psi < 0} \cdot \boldsymbol{\alpha}_\psi^\top\right) \cdot \boldsymbol{h}$
6 $\quad\quad \boldsymbol{\alpha}, lb \leftarrow \textsc{LinearBounds}(f_c, \mathcal{B}(\boldsymbol{h}_\phi, r), \phi^{cons})$
7 $\quad$ **return** $\perp$
8 **Function** `Point-wise Repair`$(f, \mathcal{P}, r, T_b, T_r)$**:**
9 $\quad C \leftarrow \{\}$
10 $\quad$ **foreach** $\phi \in \mathcal{P}$ **do**
11 $\quad\quad \triangleright\ \phi^{in}$ consists of a single point $\boldsymbol{x}_\phi$
12 $\quad\quad \boldsymbol{h}^*_\phi \leftarrow$ `Proxy Box Synthesis`$(f_c, f_e(\boldsymbol{x}_\phi), \phi, r, T_b)$
13 $\quad\quad$ **if** $\boldsymbol{h}^*_\phi = \perp$ **then return** $\perp$
14 $\quad\quad C \leftarrow C \cup \left\{\left(\boldsymbol{x}_\phi, \boldsymbol{h}^*_\phi\right)\right\}$
15 $\quad$ **for** 1 **to** T_r **do**
16 $\quad\quad$ **if** $\bigwedge_{\phi \in \mathcal{P}} f \models \phi$ **then return** $f_c \circ f_e$
17 $\quad\quad \mathcal{L} \leftarrow \frac{1}{|C|} \sum_{(\boldsymbol{x}_\phi, \boldsymbol{h}^*_\phi) \in C} \left\| f_e(\boldsymbol{x}_\phi) - \boldsymbol{h}^*_\phi \right\|_2$
18 $\quad\quad f_e \leftarrow \text{Adam}(f_e; \min \mathcal{L})$
19 $\quad$ **return** $\perp$

a general convex set defined by multiple linear constraints. This necessitates solving a linear programming for projection operations in the problem (2), which is computationally expensive.

Proxy Boxes Synthesis. To address above issues, rather than generating approximations of the whole preimage for each property $\phi \in \mathcal{P}$, we concentrate on synthesizing a proxy box $\mathcal{B}_\phi \subset f_c^{-1}(\mathbb{R}^s, \phi^{out})$. The key idea is that a small box located within the preimage and in close proximity to the original feature $f_e(\boldsymbol{x}_\phi)$ is sufficient to provide the necessary information for the repair. To obtain such box, we propose an algorithm based on linear relaxation and iterative center shifting, detailed in Algorithm 1 (lines 1-7). It starts by constructing the initial box centered around $f_e(\boldsymbol{x}_\phi)$, and then iteratively searches the current box for the point with minimal constraints violation, which then serves as the center for the next box. Specifically, for a given box centered at $\boldsymbol{h}_\phi$ with a radius r, defined as $\mathcal{B}(\boldsymbol{h}_\phi, r) = \left\{\boldsymbol{h} \in \mathbb{R}^s \mid \left\|\boldsymbol{h} - \boldsymbol{h}_{\boldsymbol{\phi}}\right\|_\infty \leq r\right\}$, we aim to find the point $\boldsymbol{h}'_\phi$ within this box that minimize the degree of constraints violation as follows:

$$\boldsymbol{h}'_\phi = \arg\min_{\boldsymbol{h} \in \mathcal{B}(\boldsymbol{h}_\phi, r)} \sum_{\psi \in \phi^{cons}} -\min\left(\boldsymbol{c}_\psi^\top \cdot f_c(\boldsymbol{h}) + d_\psi, 0\right) \tag{3}$$

However, precisely solving this optimization problem is challenging due to two inherent complexities: 1) the non-convex activation functions within f_c, and 2) the inner *min* operator). To address the former, we take the box $\mathcal{B}(\boldsymbol{h}_\phi, r)$ as the input space and conduct

linear relaxation on the subnetwork f_c. Specifically, we employ auto-LiRPA [61] to obtain the linear lower bound on the f_c's output over all constraints $\psi \in \phi^{cons}$ as follows:

$$\boldsymbol{\alpha}_\psi^\top \cdot \boldsymbol{h} + \beta_\psi \leq \boldsymbol{c}_\psi^\top \cdot f_c(\boldsymbol{h}) + d_\psi \quad \forall \boldsymbol{h} \in \mathcal{B}(\boldsymbol{h}_\phi, r),\ \psi \in \phi^{cons} \tag{4}$$

where $\boldsymbol{\alpha}_\psi \in \mathbb{R}^s$ and $\beta_\psi \in \mathbb{R}$ are the coefficients and bias. For the inner *min* operator in Eq. (3), we note that its inclusion is necessary since different points within the box may not uniformly satisfy (or violate) the constraint ψ; otherwise, the *min* function could be omitted. To simplify it, we propose a satisfaction approximation strategy. We first approximately infer whether the entire box satisfies the constraint ψ, and further generalize this inference to all points within the box. Given a constraint ψ, the property satisfaction for the entire box can be determined by minimizing the lower bound $lb_\psi = \min_{\boldsymbol{h} \in \mathcal{B}(\boldsymbol{h}_\phi, r)} \boldsymbol{\alpha}_\psi^\top \cdot \boldsymbol{h} + \beta_\psi$, where $lb_\psi \geq 0$ implies that the entire box satisfies the constraint ψ, i.e., $\min(\boldsymbol{c}_\psi^\top \cdot f_c(\boldsymbol{h}) + d_\psi, 0)$ equals 0. We filter out these already satisfied constraints and generalize the satisfaction state of the entire box to all points for the remaining constraints. Consequently, the problem Eq. (3) can be simplified to a linear optimization problem as:

$$\boldsymbol{h}'_\phi = \arg \min_{\boldsymbol{h} \in \mathcal{B}(\boldsymbol{h}_\phi, r)} - \sum_{\psi \in \phi^{cons}} \left(\boldsymbol{\alpha}_\psi^\top \cdot \boldsymbol{h} + \beta_\psi \right) \cdot \mathcal{I}_{lb_\psi < 0} \tag{5}$$

where $\mathcal{I}_{(\cdot)}$ represents the indicator function. Since the feasible region $\mathcal{B}(\boldsymbol{h}_\phi, r)$ is an axis-aligned box, problem (5) can be directly solved in a closed form. As described in Alg. 1 (lines 3-6 and 10-14), for each property $\phi \in \mathcal{P}$, we iteratively calculate $\boldsymbol{h}'_\phi$ and shift the center of the current box to $\boldsymbol{h}'_\phi$ to synthesize the next box until all constraints are satisfied within the box. This box synthesis process analyzes the model's behavior over the entire box $\mathcal{B}$ at each step, thus preventing certain feature dimensions from getting stuck in local optima where the gradients become negligible. We validate this advantage by comparing our method with gradient-based optimization and varying the radius of the box (Sec. 6.4).

Upon obtaining the proxy box $\mathcal{B}_\phi$, using it to replace $\mathcal{U}$ in Eq. (2) can bypass the costly LP solving needed for projection. However, a critical issue arises if we directly minimize the distance $\mathcal{L}_{\text{Proj}} := \|f_e(\boldsymbol{x}_\phi) - \Pi_{\mathcal{B}_\phi} f_e(\boldsymbol{x}_\phi)\|_p$: as $f_e(\boldsymbol{x}_\phi)$ approaches the surface of $\mathcal{B}_\phi$, the distance (and its gradient w.r.t. f_e) diminishes rapidly, causing the repair process to prematurely converge before $f_e(\boldsymbol{x}_\phi)$ properly enters the box. To address this issue, we exploit the proxy box's geometric nature to derive a bounded distance measure. Specifically, we establish the following theorem (proof in Appendix A.1):

Theorem 1. *Let ϕ be a point-wise property with $\phi^{in} = \{\boldsymbol{x}_\phi\}$, and $\mathcal{B}_\phi$ be the proxy box of radius r centered at $\boldsymbol{h}^*_\phi$. Given a repaired model $\tilde{f} := f_c \circ \tilde{f}_e$, the distance $\mathcal{L} := \|\tilde{f}_e(\boldsymbol{x}_\phi) - \boldsymbol{h}^*_\phi\|_p$ satisfies:*

$$\max\left(\|\boldsymbol{h}^*_\phi - \Pi_{\mathcal{B}_\phi} \tilde{f}_e(\boldsymbol{x}_\phi)\|_p ,\ \mathcal{L}_{\text{Proj}} \right) \leq \mathcal{L} \leq \mathcal{L}_{\text{Proj}} + r \cdot s^{1/p} \tag{6}$$

where $p \geq 1$ and s is the feature space dimension. We further have that if $\mathcal{L} \leq r$ holds, then $f_c(\tilde{f}_e(\boldsymbol{x}_\phi)) \in \phi^{out}$, which implies $\tilde{f} \models \phi$.

Theorem 1 states that the original objective $\mathcal{L}_{\text{Proj}}$ can be reformulated as the ℓ_p-norm distance $\mathcal{L}$, which is bounded by $\mathcal{L}_{\text{Proj}}$ and $\mathcal{L}_{\text{Proj}} + r \cdot s^{1/p}$. Crucially, before $f_e(\boldsymbol{x}_\phi)$ enters $\mathcal{B}_\phi$, the inequality $r \leq \|\boldsymbol{h}^*_\phi - \Pi_{\mathcal{B}_\phi} \tilde{f}_e(\boldsymbol{x}_\phi)\|_p \leq \mathcal{L}$ always holds, thereby avoiding the premature convergence issue. The overall algorithm for point-wise repair is summarized in Alg. 1. For all properties $\phi \in \mathcal{P}$, we synthesize the proxy box and optimize f_e to minimize $\mathcal{L}$ until all properties are satisfied. For simplicity, we use the Euclidean distance ($p = 2$) in our implementation. Notably, the calculation of the proxy box for each property (lines 10-14 of Alg. 1) can be parallelized.

5.2 Region-wise Repair

Now we present how ProRepair handles region-wise repair, which is more challenging than point-wise task. The core difficulty stems from the input space ϕ^{in} consisting of an infinite number of points, making it necessary to characterize the model's behavior across the entire space and perform targeted adjustments. For this, prior works combine the activation status with constraint solving, e.g., enumerating all vertices of ϕ^{in} and enforcing the model to exhibit identical activation status on all vertices to induce linear behavior throughout the space. Based on this idea, the state-of-the-art method APRNN can handle spaces up to 16 dimensions (with 2^{16} vertices). However, such activation-driven methods suffer from two inherent limitations: the exponential complexity of vertex enumeration confines the scalability to higher dimensional spaces, and the rigid requirement of uniform activation status across the entire space severely limits the feasible solution space.

To tackle these problems, we transform the region-wise repair into an iterative process, where we let counterexample generation take the lead, and employ property refinement for more surgical repair. The core insight is that correcting the model's erroneous output at point $\boldsymbol{x}$ could simultaneously rectify erroneous behaviors over the neighborhood due to the model's continuity.

Counterexample Generation. Following the above idea, the first step is to identify a counterexample (an input point $\boldsymbol{x} \in \phi^{in}$ that $f(\boldsymbol{x}) \notin \phi^{out}$) for each property. Intuitively, repairing the input point with the most constraints violation within ϕ^{in} is more effective in correcting the model's erroneous behavior over the entire space. Inspired by this, we first identify the "worst" input $\boldsymbol{x}_\phi$ for each property ϕ, which can be formalized as:

$$\boldsymbol{x}_\phi = \arg \max_{\boldsymbol{x} \in \phi^{in}} \sum_{\psi \in \phi^{cons}} - \min\left(\boldsymbol{c}_\psi^\top \cdot f(\boldsymbol{x}) + d_\psi, 0 \right) \tag{7}$$

We note that problem (7) has the same structure as Eq. (3) except for the inverted objective and different feasible space. Thus, we conduct linear relaxation on the whole model f with the space ϕ^{in} to derive the approximation of the objective in Eq. (7):

$$\begin{aligned} &\sum_{\psi \in \phi^{cons}} - \min\left(\boldsymbol{c}_\psi^\top f(\boldsymbol{x}) + d_\psi, 0 \right) \\ \leq &\sum_{\psi \in \phi^{cons}} - \min\left(\underline{\boldsymbol{w}}_\psi^\top \boldsymbol{x} + b_\psi, 0 \right) = \sum_{\psi \in \text{Vio}(\phi)} - \min\left(\underline{\boldsymbol{w}}_\psi^\top \boldsymbol{x} + b_\psi, 0 \right) \end{aligned} \tag{8}$$

where $\underline{\boldsymbol{w}}_\psi \in \mathbb{R}^m$ and $\underline{b}_\psi \in \mathbb{R}$ are the coefficients and bias computed via auto-LiRPA such that $\underline{\boldsymbol{w}}_\psi^\top \cdot \boldsymbol{x} + \underline{b}_\psi \leq \boldsymbol{c}_\psi^\top \cdot f(\boldsymbol{x}) + d_\psi$ holds, $\text{Vio}(\phi)$ denotes the set of constraints that may not be satisfied, i.e., $\{\psi \in \phi^{cons} \mid lb_\psi = \min_{\boldsymbol{x} \in \phi^{in}} \underline{\boldsymbol{w}}_\psi^\top \cdot \boldsymbol{x} + \underline{b}_\psi < 0\}$. We then utilize the satisfaction approximation strategy proposed in Sec. 5.1 to linearize the objective as $\max_{\boldsymbol{x} \in \phi^{in}} \sum_{\psi \in \text{Vio}(\phi)} - \left(\underline{\boldsymbol{w}}_\psi^\top \boldsymbol{x} + b_\psi \right)$. The above simplification yields a linear objective, enabling us to compute the

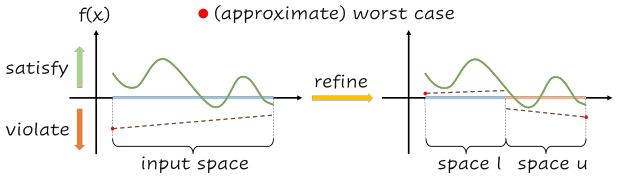


Figure 3: Illustration of refinement: the approximate input does not violate constraints before refinement, while a valid counterexample is captured by splitting the input space.

optimal solution directly. After generating the counterexample for each property, we collect all the real counterexamples and formulate a point-wise repair task, which can be handled by Alg. 1.

However, the above counterexample generation process might not always produce valid counterexamples. This issue arises from the potentially large input space ϕ^{in} (high-dimensional with wide-ranging values), which can lead to excessive approximation errors in Eq. (8), especially since, unlike the proxy box synthesis step, it needs to approximate a more complex model (f rather than f_c).

Property Refinement. To mitigate this problem, we propose to refine the unsatisfied properties after one iteration point-wise repair. As illustrated in Fig. 3, through partitioning the input space into subspaces, we aim to more precisely capture the model's outputs, thereby facilitating counterexample generation and enabling more targeted model repair. Note that splitting all dimensions of the input space is impractical, as a single refinement step can result in an exponential number of sub-properties. Hence, selecting the appropriate dimensions to split is crucial. To this end, we devise a significance metric called *Refine Score* (RS) to assess the splitting importance of each dimension i, considering both its input range magnitude and impact on the model's output:

$$\mathrm{RS}(\phi)_i = \left(\boldsymbol{u}_i^{\phi} - \boldsymbol{l}_i^{\phi}\right) \odot \sum_{\psi \in \mathrm{Vio}(\phi)} \left|\underline{\boldsymbol{w}}_{\psi}\right|_i,\ i \in [1, m] \tag{9}$$

where $\odot$ is the element-wise multiplication, and $\sum_{\psi \in \mathrm{Vio}(\phi)} |\underline{\boldsymbol{w}}_{\psi}|_i$ represents the estimated impact of the dimension i on the model's output (considering the unsatisfied constraints). We select the dimension with the maximum RS value to bisect the input space ϕ^{in} into ϕ_l^{in} and ϕ_u^{in}.

The main loop of our region-wise repair method is summarized in lines 20-28 of Algorithm 2, which is coupled with two key functions: counterexample generation (lines 1-10) and property refinement (lines 11-19). It first generates the counterexamples and then constructs a point-wise task, followed by a property refinement step. When the set $\mathrm{Vio}(\phi)$ for all properties is empty, we return a repaired model. Otherwise, the algorithm continues to iterate until the property budget $\mathbf{T}$ is exhausted. The following theorem shows the soundness of our method (proof in Appendix A.2).

Theorem 2. *Given a property set $\mathcal{P}$ and a model f, Algorithm 2 returns a repaired model $\tilde{f}$ only if $\tilde{f} \models \mathcal{P}$ holds.*

Asymptotic complexity analysis. The core components of ProRepair include box synthesis, repair, counterexample generation and space refinement. Let the respective costs be denoted as C_{box}, C_{repair}, C_{counter}, and C_{refine}. With T_b, T_r and $\mathbf{T}$ denoting the iteration budgets for box synthesis, repair and refinement, and assuming each layer contains at most K neurons, the costs of these components are shown in Tab. 2. The overall cost is $\mathbf{T} * (T_b * L_c^2 * K^3 + T_r * L_e^2 * K^3 + (L_e + L_c)^2 * K^3 + m * q)$. Ignoring constant factors, this can be simplified to $O((L_e + L_c)^2 * K^3)$, which grows quadratically with the total number of layers and cubically with the maximum number of neurons per layer.

Algorithm 2: Region-wise Repair

Input: DNN $f = f_c \circ f_e$, property set $\mathcal{P}$, box radius r and property budget $\mathbf{T} = 10000$
Output: repaired model $\tilde{f}$ that $\tilde{f} \models \mathcal{P}$, or $\perp$

1 **Function** `Counterexample Generation`$(f, \mathcal{P})$:
2 $\quad \mathcal{Q} \leftarrow \{\}$ ▷ Store generated point-wise properties
3 $\quad$ **foreach** $\phi \in \mathcal{P}$ **do**
4 $\quad\quad \underline{\boldsymbol{w}}, lb \leftarrow \textsc{LinearBounds}\left(f, \phi^{in}, \phi^{cons}\right)$
5 $\quad\quad \mathrm{Vio}(\phi) \leftarrow \left\{\psi \in \phi^{cons} \mid lb_{\psi} < 0\right\}$
6 $\quad\quad \boldsymbol{x}_{\phi} \leftarrow \arg\min_{\boldsymbol{x} \in \phi^{in}} \sum_{\psi \in \mathrm{Vio}(\phi)} \boldsymbol{w}_{\psi}^{\top} \cdot \boldsymbol{x}$
7 $\quad\quad$ **if** $f(\boldsymbol{x}_{\phi}) \notin \phi^{out}$ **then**
8 $\quad\quad\quad \mathcal{Q} \leftarrow \mathcal{Q} \cup \left\{\left(\{\boldsymbol{x}_{\phi}\}, \phi^{cons}, \phi^{out}\right)\right\}$
9 $\quad\quad \mathrm{RS}(\phi) \leftarrow \left(\boldsymbol{u}^{\phi} - \boldsymbol{l}^{\phi}\right)^{\top} \odot \sum_{\psi \in \mathrm{Vio}(\phi)} \left|\boldsymbol{w}_{\psi}\right|$ ▷ $\mathrm{RS}(\phi) \in \mathbb{R}^m$
10 $\quad$ **return** Vio, $\mathcal{Q}$, RS
11 **Function** `Property Refinement`$(\mathcal{P}, \mathrm{Vio}, \mathrm{RS})$:
12 $\quad \mathcal{P}' \leftarrow \{\phi \in \mathcal{P} \mid \mathrm{Vio}(\phi) = \emptyset\}$
13 $\quad$ **foreach** $\phi \in \{\phi \mid \phi \in \mathcal{P} \wedge \mathrm{Vio}(\phi) \neq \emptyset\}$ **do**
14 $\quad\quad d \leftarrow \arg\max_{1 \le i \le m} \mathrm{RS}(\phi)_i$
15 $\quad\quad \phi_l \leftarrow \phi,\ \phi_l^{in} \leftarrow \phi^{in} \cap \left\{\boldsymbol{x} \mid \boldsymbol{x}_d \le (\boldsymbol{l}_d^{\phi} + \boldsymbol{u}_d^{\phi})/2\right\}$
16 $\quad\quad \phi_u \leftarrow \phi,\ \phi_u^{in} \leftarrow \phi^{in} \cap \left\{\boldsymbol{x} \mid \boldsymbol{x}_d \ge (\boldsymbol{l}_d^{\phi} + \boldsymbol{u}_d^{\phi})/2\right\}$
17 $\quad\quad \mathcal{P}' \leftarrow \mathcal{P}' \cup \{\phi_l,\ \phi_u\}$
18 $\quad\quad \mathrm{Vio}(\phi_l) \leftarrow \mathrm{Vio}(\phi),\ \mathrm{Vio}(\phi_u) \leftarrow \mathrm{Vio}(\phi)$
19 $\quad$ **return** $\mathcal{P}'$
20 **Function** `Region-wise Repair`$(f, \mathcal{P}, r)$:
21 $\quad \mathrm{Vio}, \mathcal{Q}, \mathrm{RS} \leftarrow$ `Counterexample Generation`$(f, \mathcal{P})$
22 $\quad$ **while** $|\mathcal{P}| \le \mathbf{T}$ **do**
23 $\quad\quad f \leftarrow \textsc{Point-Wise Repair}(f, \mathcal{Q}, r, T_b, T_r)$ ▷ Alg. 1
24 $\quad\quad$ **if** $f = \perp$ **then return** $\perp$
25 $\quad\quad \mathrm{Vio}, \mathcal{Q}, \mathrm{RS} \leftarrow$ `Counterexample Generation`$(f, \mathcal{P})$
26 $\quad\quad$ **if** $\bigwedge_{\phi \in \mathcal{P}} (\mathrm{Vio}(\phi) = \emptyset)$ **then return** f
27 $\quad\quad \mathcal{P} \leftarrow$ `Property Refinement`$(\mathcal{P}, \mathrm{Vio}, \mathrm{RS})$
28 $\quad$ **return** $\perp$

6 Evaluation

We apply ProRepair to four different repair tasks, including point-wise repair for natural corruption and backdoor attacks, as well as region-wise repair for adversarial attacks and violation of safety properties. We briefly introduce the setup below. More details on

Table 2: Complexity of each component in ProRepair.

	Description	Cost
C_{box}	Bound propagation over f_c across L_c layers, where each neuron contributes up to $L_c * K^2$ terms	$T_b * L_c^2 * K^3$
C_{repair}	Gradient descent optimization on f_e ($L_e * K$ neurons with $L_e * K^2$ terms per neuron)	$T_r * L_e^2 * K^3$
C_{counter}	Bound propagation on the model f	$(L_e + L_c)^2 * K^3$
C_{refine}	Refinement score computation over an m-D space with q constraints	$m * q$

the models, datasets, and the construction of the desired property set are provided in Appendix B.

6.1 Experimental Setup

6.1.1 Repair Task.

- **Backdoor.** This task aims to repair the buggy NN so that it correctly classify the backdoored data, which is stronger than merely defending against attacks (i.e., ensuring the classification is not the target label). We conduct extensive evaluations on multiple datasets, attack strategies, and model architectures. More setup for this task are detailed in the Appendix B.1.1 and Appendix B.3.
- **Corruption.** This task is widely adopted in provable repair [13, 47, 50] to evaluate the effectiveness and efficiency of repair methods on small-scale datasets and models. We use various NNs from [43] on MNIST-C [39] dataset with all corruptions.
- **Adversarial attack.** This task aims to correct a buggy model so that it is provably robust within the given local spaces. We conduct evaluations on the MNIST, GTRSB and CIFAR-10 datasets. The corresponding networks are 3x100, CNN-3 and CNN-4. We evaluate our method and APRNN by varying the perturbation dimensions and the number of desired properties.
- **Safety Property Violation.** This task focuses on correcting the network so that it satisfies the property that may be defined on a global input space rather than a local neighborhood. Specifically, we conduct the evaluation on the ACAS Xu benchmark that is designed for aircraft collision detection. As reported in [10], more than 30 ACAS Xu models violate Property-2 (detailed in Appendix B.1.4). We perform an evaluation of all these models.

6.1.2 Baselines and Configurations. To the best of our knowledge, APRNN, PRDNN, and REASSURE are the only three provable repair methods that scale to medium-sized and larger DNNs. We conduct the evaluation by comparing our method with these SOTA methods. For the backdoor repair task, we include two additional methods: SEAM [71] and I-REPAIR [22]. SEAM is the latest SOTA unlearning method for backdoor removal, while I-REPAIR is a repair framework based on fault localization and fine-tune. They do not provide guarantees and require additional clean data, but typically exhibit notable scalability and efficiency.

For ProRepair, we set the radius r of the box to 0.1 for all scenarios, except for backdoor repair, where it is set to 0.5 due to the larger model size. We analyze the impact of this parameter in Sec. 6.4. Other setups for all methods are detailed in Appendix B.4.

6.1.3 Evaluation metrics. There are three metrics involved in the evaluation: *PSR* (Property Satisfaction Rate), *Acc* (Accuracy), and *Gene* (Generalization). *PSR* measures the proportion of given desired properties that are satisfied by the repaired model, i.e., PSR = $\frac{|\{\phi \in \mathcal{P} | \tilde{f} \models \phi\}|}{|\mathcal{P}|}$, For provable repairs, the *PSR* should reach 100%. *Acc* measures the ratio of normal inputs from the test set that can be correctly classified. *Gene* measures the proportion of inputs from the generalization set for which the model's outputs satisfy the desired properties. Details on the construction of the desired property set $\mathcal{P}$ and the generalization set are provided in Appendix B.

6.2 How effective and efficient is ProRepair?

6.2.1 Point-wise Backdoor Repair. Considering the limited availability of poisoned data in practice, we randomly selected 50 buggy data (nearly 0.1% of the training dataset) to construct the desired property set. We first conducted evaluations under relatively common attack strategies, i.e., dirty label attack on single class. REASSURE was excluded from the evaluation, as it does not directly support such large scale models. For SEAM and I-REPAIR, we provide an additional clean set containing 1% of the training data.

As shown in Tab. 3, both PRDNN and APRNN achieve provable repair (i.e., 100% PSR). Nevertheless, they encounter failures in some cases: PRDNN requires significant memory, leading to out-of-memory issues, while APRNN encountered some unsolvable instances, as the constructed LP has no feasible solution. This is because they differ from data-driven approaches, as more properties to repair introduces additional constraints, making the task more challenging. By comparison, ProRepair successfully corrects the model across all settings. In Sec. 6.3, we conduct further evaluation with varying amounts of properties to repair.

In terms of accuracy preservation and generalization, ProRepair outperforms in the majority of settings: the average accuracy drop and generalization are below 2% and about 70%, respectively. In contrast, the baseline with the best accuracy preservation, I-REPAIR, shows a performance loss of about 6%, while SEAM exhibits the best generalization among all baselines, at around 35%. PRDNN and APRNN do not consistently protect model performance: they only perform well under few settings. Their generalization is also unsatisfactory, possibly because they require that activation patterns remain unchanged before and after repair, as some studies [32, 69] reveal that backdoored models often show unusually high activation in certain neurons on poisoned samples. On the other hand, the performance of SEAM and I-REPAIR is highly influenced by the dataset, due to their data-driven nature. On more complex datasets (e.g., CIFAR-100), they exhibit limited generalization and significant accuracy degradation, as these more complex scenarios require larger amounts of data. We evaluate their performance under varying sample (including both buggy and clean data) sizes in Sec. 6.3. Nevertheless, they still demonstrate decent efficiency. The other two provable repair methods often require several hundred to thousands of seconds, with PRDNN's overheads in some scenarios exceeding 10 000 seconds, even surpassing the time needed to train a model from scratch. Overall, our method performs the best in

Table 3: Point-wise backdoor repair on different models, datasets, and attacks. The best values are in bold, and the second-best values are underlined. PSR: Property Satisfaction Rate (%), Acc: Accuracy (%), Gene: Generalization (%), T: Time (seconds).

		Attack	Original		ProRepair				PRDNN[47]				APRNN[50]				SEAM[71]				I-REPAIR[22]			
			Acc	Gene	PSR	Acc	Gene	T	PSR	Acc	Gene	T	PSR	Acc	Gene	T	PSR	Acc	Gene	T	PSR	Acc	Gene	T
ResNet18	SVHN	BNA2O	95.9	8.4	**100**	**94.91**	**94.63**	**1.2**	**100**	94.60	19.51	4107	**100**	94.37	18.84	913	96	62.91	58.81	5.1	74	94.53	52.68	**20.2**
		Blend	95.8	6.4	**100**	94.13	**78.57**	**1.9**	**100**	83.01	6.84	2822	**100**	85.64	8.90	1028	90	71.12	55.65	2.2	0	**94.48**	6.36	**24.9**
		TrojanNN	96.1	6.7	**100**	**95.27**	**72.76**	**1.2**	**100**	84.58	9.51	3549	**100**	80.49	12.60	1198	84	68.05	34.08	2.6	18	92.88	7.61	**23.5**
	GTSRB	BNA2O	98.6	7.4	**100**	**98.55**	**98.44**	**1.2**	**100**	84.60	23.02	23059	**100**	92.41	18.97	1313	98	80.09	79.31	1.9	84	97.19	51.11	**3.8**
		Blend	98.5	1.2	**100**	**97.16**	**89.77**	**1.3**	**100**	62.00	9.83	19076	**100**	58.45	6.55	1409	94	77.82	59.21	1.9	20	93.02	7.17	**4.1**
		TrojanNN	98.9	0.5	**100**	**97.60**	**91.94**	**1.2**	**100**	64.39	9.24	20162	**100**	77.12	12.79	1657	82	80.57	20.19	1.4	92	91.17	39.48	**3.4**
	CIFAR10	BNA2O	93.6	12.1	**100**	90.50	**89.84**	**1.2**	**100**	92.44	36.58	5617	**100**	**92.87**	30.20	1748	96	44.04	41.27	**6.6**	78	84.81	57.02	3.6
		Blend	94.8	10.0	**100**	**90.41**	**67.98**	**1.3**	**100**	80.72	16.44	3691	**100**	77.11	16.67	672	82	65.49	28.71	1.5	0	88.05	10.00	**4.2**
		TrojanNN	94.4	10.0	**100**	92.13	**62.36**	**1.3**	**100**	91.50	18.70	6419	**100**	**92.76**	15.04	919	50	60.22	21.24	1.4	0	92.48	10.04	**4.3**
	CIFAR100	BNA2O	76.0	5.5	**100**	**73.50**	**64.26**	**1.6**	Out of Memory[1]				**100**	73.10	8.06	897	92	8.05	7.39	**6.7**	62	62.20	12.01	3.9
		Blend	75.5	1.0	**100**	**70.57**	**32.79**	**1.8**					**100**	58.75	2.11	814	72	8.11	5.14	**6.0**	4	63.94	1.19	4.3
		TrojanNN	75.7	1.3	**100**	**73.40**	**49.38**	**1.6**					**100**	63.83	2.69	902	70	7.16	3.64	**4.7**	24	65.34	2.80	4.3
VGG11[2]	SVHN	BNA2O	95.2	8.7	**100**	**94.68**	**93.86**	**1.4**	**100**	42.19	38.08	1527	**100**	50.46	37.06	142	92	69.69	68.16	**3.3**	12	94.52	13.83	2.2
		Blend	94.8	6.4	**100**	**94.23**	**82.90**	**1.6**	**100**	50.39	34.88	2314	**100**	32.37	41.25	161	96	69.50	63.77	**3.4**	36	89.35	23.88	2.3
		TrojanNN	95.2	6.7	**100**	**94.54**	**72.94**	1.3	**100**	48.03	17.23	1017	Repair Failed				60	79.84	27.92	**0.9**	12	93.65	7.32	**2.5**
	GTSRB	BNA2O	97.5	7.6	**100**	**97.32**	**95.15**	1.4	**100**	23.03	15.60	5765	**100**	19.79	18.28	219	96	64.12	50.32	**1.6**	**100**	96.84	67.09	**1.2**
		Blend	97.6	1.2	**100**	**96.70**	**88.83**	1.5	**100**	8.42	10.57	6018	**100**	5.42	11.75	342	90	80.48	55.51	**0.8**	6	96.55	1.90	**2.1**
		TrojanNN	97.6	0.6	**100**	96.13	**73.83**	1.4	**100**	16.72	10.73	6420	**100**	8.76	11.55	665	84	81.92	36.06	**0.4**	98	**96.32**	42.73	**1.5**
	CIFAR10	BNA2O	91.5	12.4	**100**	**89.68**	**87.18**	**2.3**	**100**	36.56	35.98	1016	Repair Failed				70	66.51	55.32	**0.4**	2	87.11	13.36	1.4
		Blend	91.5	10.0	**100**	**88.76**	**74.30**	**2.1**	**100**	13.78	22.71	875	**100**	42.57	23.90	173	86	65.99	29.34	**0.7**	0	85.76	10.00	1.8
		TrojanNN	91.7	10.0	**100**	**89.79**	**54.72**	1.4	**100**	23.79	20.08	1955	**100**	52.70	24.54	393	72	57.85	28.33	**1.2**	0	88.87	10.07	**2.2**
	CIFAR100	BNA2O	70.7	5.9	**100**	**67.62**	**61.84**	2.1	**100**	16.01	13.34	13956	**100**	14.31	15.69	271	24	26.63	8.56	**1.1**	58	49.47	13.74	**2.6**
		Blend	71.8	1.0	**100**	**69.04**	**29.46**	1.6	**100**	14.63	4.80	15797	**100**	9.03	5.92	253	34	40.49	3.32	**0.6**	18	43.34	2.61	**2.9**
		TrojanNN	71.1	1.2	**100**	**68.01**	**38.99**	1.7	**100**	14.02	5.73	14592	**100**	16.06	6.54	281	12	37.12	3.10	**0.4**	38	49.80	4.07	**2.6**

1. PRDNN required more than 256GB of memory in these settings, resulting in out-of-memory errors. 2. We employed VGG16 for CIFAR-100 and VGG11 for other datasets.

Table 4: Backdoor repair against advanced attacks.

	Method	Badnets All to All				SIG			
		PSR	Acc	Gene	T	PSR	Acc	Gene	T
ResNet18	Original	0	94.94	2.97	-	0	98.72	0.73	-
	PRDNN	98[1]	94.83	25.18	5443	**100**	37.32	6.72	16683
	APRNN	**100**	**94.87**	15.61	1144	**100**	32.23	9.36	1426
	SEAM	88	52.86	45.21	2.5	84	72.01	32.66	2.9
	IREPAIR	**100**	93.14	77.63	6.3	60	71.16	33.48	3.8
	Ours	**100**	93.41	**86.74**	**0.7**	**100**	**94.05**	**69.50**	**1.5**
VGG11	Original	0	91.87	4.18	-	0	97.83	3.42	-
	PRDNN	**100**	**91.59**	32.96	1008	**100**	12.42	18.58	6071
	APRNN	Repair Failed				**100**	19.07	17.32	196.2
	SEAM	74	72.29	65.97	**0.8**	60	86.30	31.03	**0.8**
	IREPAIR	52	90.97	51.51	6.4	70	93.44	39.55	1.9
	Ours	**100**	90.06	**74.07**	1.0	**100**	**96.59**	**59.75**	1.1

this regard, with repair overheads never exceeding 2 seconds (a speed-up of 160× to 2000× over existing provable methods).

Against advanced attack. We further consider two more advanced attacks: Badnets All-to-All and SIG on CIFAR-10 and GTSRB, respectively. The former involves dirty label attacks across multiple classes, while the latter is an invisible clean label attack. The desired property set is constructed from 50 randomly selected buggy data. As shown in the Tab. 4, our method remains robust, with an average accuracy drop of only 2.3%, and a generalization improvement that exceeds 70%. By comparison, other baselines exhibit unstable results. While I-REPAIR performs better than the others, it still falls significantly behind our method in terms of generalization and accuracy preservation.

6.2.2 Point-wise Corruption Repair. The desired property set for this task is constructed using 100 corrupted inputs from the repair set. We consider all 15 types of corruption from the MNIST-C dataset and report the full results for the 6x100 model in Tab. 5. Results for the remaining models are shown in Fig. 9 in the Appendix.

Considering all corruption types and model architectures, most methods achieved provable repair (i.e., 100% PSR) in the majority of settings. However, APRNN failed to find feasible solution in certain cases (e.g., 9x200 model with *scale* corruption). In terms of accuracy, APRNN and PRDNN showed varying degrees of drop: APRNN decreased by an average of 8.97%, while PRDNN decreased by 10.39%. In comparison, our method exhibited a much smaller average decrease of only 1.76%. On the other hand, REASSURE did not show any decline, as it constructs a patch network and a support network for each buggy data, which essentially acts as a look-up table. However, this also leads to a nearly 0% generalization. PRDNN and APRNN achieved better generalization of 67.2% and 71.8%, respectively. Our method consistently achieved the highest generalization across all settings, with an average of 85%. The improvement in generalization is especially significant under challenging corruptions such as *fog* and *brightness*. For simpler corruptions, the gain is naturally smaller, as the models already exhibit strong generalization, sometimes approaching the accuracy on clean samples.

Time Cost. REASSURE had a significant time overhead (more than 4 000 seconds on average) due to its need to calculate the activation patterns and linear regions of the network for each buggy data, resulting in a substantial increase in time cost as the number of neurons increases. The average repair times for APRNN and PRDNN were 269s and 9s, respectively. Our method demonstrated the best

Table 5: Results of point-wise corruption repair on the 6×100 model under different corruptions.

Corruption	Original		ProRepair				PRDNN[47]				APRNN[50]				REASSURE†[13]			
	Acc	Gene	PSR	Acc	Gene	T	PSR	Acc	Gene	T	PSR	Acc	Gene	T	PSR	Acc	Gene	T
brightness	96.29	39.89	**100**	95.31	**83.49**	**1.5**	**100**	94.84	45.31	4.4	**100**	93.60	65.78	144.4	**100**	**96.29**	39.89	1498.1
canny edges	96.29	61.10	**100**	94.93	**72.50**	**1.5**	**100**	92.54	61.62	5.4	**100**	93.32	65.02	133.8	**100**	**96.29**	61.10	2056.0
dotted line	96.29	94.21	**100**	**96.58**	**94.67**	**1.6**	**100**	95.56	93.49	4.8	**100**	95.37	93.02	12.3	**100**	96.29	94.52	1857.6
fog	96.29	23.20	**100**	94.54	**81.93**	**1.4**	**100**	89.41	30.03	5.3	**100**	93.65	56.21	158.6	**100**	**96.29**	23.20	2115.3
glass blur	96.29	92.67	**100**	94.42	**92.88**	**1.6**	**100**	94.77	91.22	5.0	**100**	93.66	90.90	131.5	**100**	**96.29**	92.69	1805.6
identity	96.29	96.32	**100**	**97.27**	**97.06**	**1.9**	**100**	96.39	96.21	4.9	**100**	96.74	96.58	113.0	**100**	96.29	96.99	1845.3
impulse noise	96.29	88.07	**100**	96.20	**90.57**	**1.5**	**100**	95.40	87.81	4.8	**100**	90.86	84.12	124.8	**100**	**96.29**	88.07	1895.1
motion blur	96.29	73.41	**100**	93.71	**87.76**	**1.6**	**100**	88.31	73.91	5.4	**100**	91.24	78.59	119.0	**100**	**96.29**	73.41	2047.1
rotate	96.29	81.41	**100**	95.71	**85.41**	**1.6**	**100**	94.22	82.08	5.2	**100**	90.60	77.77	14.5	**100**	**96.29**	81.41	1971.8
scale	96.29	60.18	**100**	94.14	**85.17**	**1.6**	**100**	90.55	68.53	5.7	**100**	91.81	71.02	143.0	**100**	**96.29**	60.18	2102.3
shear	96.29	91.07	**100**	95.41	**91.48**	**1.9**	**100**	95.45	90.19	5.3	**100**	91.70	86.12	140.6	**100**	**96.29**	91.07	1967.3
shot noise	96.29	95.20	**100**	**97.14**	**95.71**	**1.7**	**100**	95.70	94.74	5.2	**100**	95.69	94.04	14.4	**100**	96.29	95.67	2020.5
spatter	96.29	94.03	**100**	**97.03**	**94.77**	**1.6**	**100**	94.84	92.16	5.2	**100**	96.03	93.88	13.9	**100**	96.29	94.51	1907.4
stripe	96.29	54.46	**100**	95.76	**78.72**	**1.3**	**100**	94.99	60.12	5.4	**100**	92.73	67.44	97.4	**100**	**96.29**	54.46	1871.8
zigzag	96.29	83.64	**100**	95.78	**89.12**	**1.7**	**100**	93.46	83.27	4.9	**100**	94.53	83.23	135.6	**100**	**96.29**	83.64	1855.7
Average	96.29	75.26	**100**	95.60	**88.08**	**1.6**	**100**	93.76	76.71	5.1	**100**	93.44	80.25	99.79	**100**	96.29	75.39	1921.1

efficiency with an average repair time of 1.8s, outperforming existing provable methods by 5× to 2000× in speed.

Remark 1: Compared to state-of-the-art repair methods, ProRepair demonstrates superior effectiveness and efficiency (with **5×-2000× speedup** against provable repair techniques) while maintaining model accuracy in both point-wise corruption and backdoor repair tasks across various settings.

6.2.3 Region-wise Adversarial Attack repair. We compare our method with APRNN, the SOTA region-wise repair approach. Here, we focus on repairing a single local space ($|\mathcal{P}| = 1$), same as the setup in APRNN (repairing with multiple properties is evaluated in Sec. 6.3).

Fig. 4 shows that for MNIST, both APRNN and ProRepair performed well when the dimensionality of the input space is less than 10, with a maximum accuracy drop of 1.28% and 0.41%, respectively. However, as dimensionality increases, APRNN exhibits significant accuracy degradation, e.g., with a 16D space, the maximum accuracy drop reaches 20.13%. Additionally, among the 20 repair tasks, there was one instance of solving failure due to an infeasible LP and one instance of memory out. This is because for the region-wise task, it enumerates the vertices of the input space, whose number grows exponentially with the dimensionality (e.g., 2^{16} vertices for the 16D space). This overwhelming increase necessitates excessive modifications to the model parameters, leading to poor performance and scalability, and even infeasible instances. The time costs also increase exponentially, averaging over 12 000 seconds. By comparison, our method demonstrates significantly better scalability, consistently maintaining accuracy and completing repairs in less than 1 second. On more complex datasets like CIFAR-10 and GTSRB, APRNN struggled further, with repairs failing at higher dimensions and accuracy drops of 7.99% and 8.70% at 8 and 9 dimensions, respectively. In comparison, ProRepair leveraged NN verification to capture the model's outputs, thereby avoiding the constraints on activation status at all vertices, which enhances scalability. It consistently completes repairs in less than 1 second for CIFAR-10 (1.08% drop) and in 3 seconds for GTSRB (2% drop), outperforming APRNN in both time cost and accuracy retention.

Table 6: Results of global safety property repair. #P represents the number of sub-properties contained in the desired property set $\mathcal{P}$ after repair.

	APRNN	Time	Ours	#P	Time		APRNN	Time	Ours	#P	Time
$N_{2,1}$	Fail	-	Succ	112	**2.99**	$N_{2,2}$	Fail	-	Succ	114	**3.01**
$N_{2,3}$	Fail	-	Succ	119	**3.86**	$N_{2,4}$	Succ	22.97	Succ	108	**3.18**
$N_{2,5}$	Fail	-	Succ	235	**4.14**	$N_{2,6}$	Fail	-	Succ	22	**2.26**
$N_{2,7}$	Fail	-	Succ	114	**15.91**	$N_{2,8}$	Fail	-	Succ	202	**3.42**
$N_{2,9}$	Fail	-	Succ	424	**7.80**	$N_{3,1}$	Succ	19.66	Succ	268	**3.87**
$N_{3,2}$	Fail	-	Succ	129	**3.01**	$N_{3,3}$	Succ	26.05	Succ	102	**3.16**
$N_{3,4}$	Fail	-	Succ	32	**0.90**	$N_{3,5}$	Fail	-	Succ	134	**3.83**
$N_{3,6}$	Fail	-	Succ	126	**3.82**	$N_{3,7}$	Fail	-	Succ	105	**2.89**
$N_{3,8}$	Fail	-	Succ	142	**3.57**	$N_{3,9}$	Fail	-	Succ	445	**9.78**
$N_{4,1}$	Succ	20.67	Succ	139	**3.24**	$N_{4,2}$	Fail	-	Succ	111	**3.33**
$N_{4,3}$	Fail	-	Succ	110	**4.16**	$N_{4,4}$	Fail	-	Succ	105	**3.48**
$N_{4,5}$	Fail	-	Succ	244	**3.69**	$N_{4,6}$	Fail	-	Succ	117	**3.52**
$N_{4,7}$	Fail	-	Succ	112	**4.15**	$N_{4,8}$	Succ	22.66	Succ	219	**3.42**
$N_{4,9}$	Succ	22.90	Succ	413	**20.94**	$N_{5,1}$	Fail	-	Succ	111	**3.37**
$N_{5,2}$	Succ	21.51	Succ	62	**1.17**	$N_{5,3}$	Succ	24.65	Succ	37	**0.94**
$N_{5,4}$	Fail	-	Succ	125	**4.13**	$N_{5,5}$	Fail	-	Succ	112	**3.47**
$N_{5,6}$	Fail	-	Succ	120	**3.07**	$N_{5,7}$	Fail	-	Succ	101	**3.46**
$N_{5,8}$	Fail	-	Succ	115	**3.03**	$N_{5,9}$	Fail	-	Succ	115	**4.67**

6.2.4 Region-wise Global Safety Property Repair. This task aims to repair a network so that the global property is satisfied. Note that non-complete verifiers face challenges with certain hard cases in verifying these global properties. Thus, we invoke a complete verifier [57] whenever the total number of sub-properties doubles, starting with an initial threshold of 100. We compare APRNN with our method in terms of the number of successful repairs and repair costs. We also record the number of sub-properties refined during the repair process, as the training and test sets of the ACAS Xu network are not publicly available. Notably, APRNN manually partitions the space into more than 500 subspaces before repair, as enforcing consistent activation status across all vertices in the

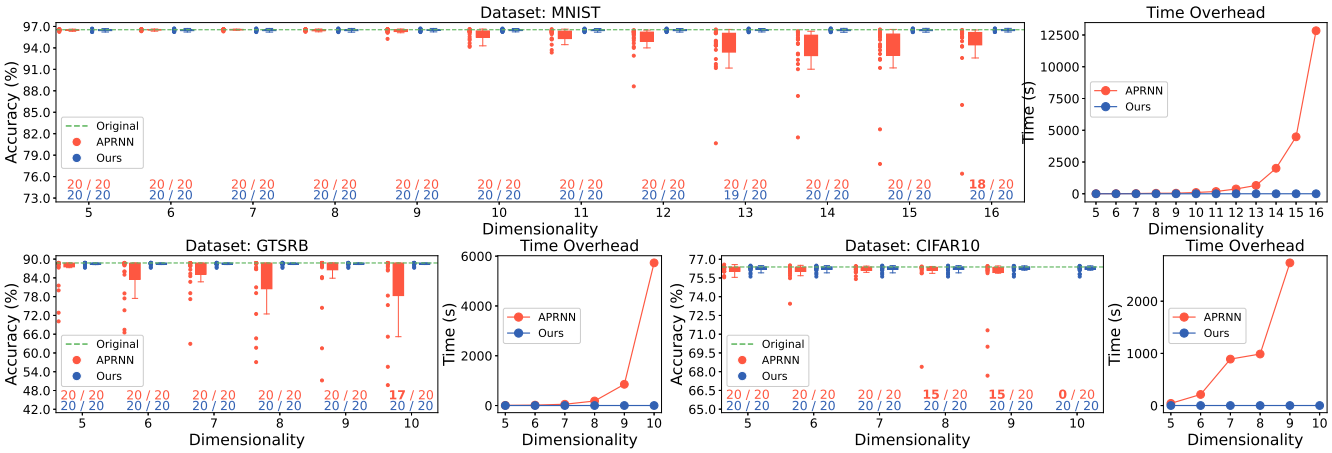


Figure 4: Results of adversarial attack repair. The x-axis represents the dimensionality of the space defined in desired properties. The scatter points on box sides represent the accuracy after each repair task. We conducted the experiments 20 times, each with randomly selected data to construct the desired properties. Values above the x-axis show the number of successful/total repairs, with red for APRNN and blue for ProRepair.

global space is impractical. The results shown in Tab. 6 highlight the remarkable improvement on repair capability of our method, as it successfully repairs all 36 instances, significantly outperforming APRNN, which only managed to repair 8 models. Moreover, ProRepair demonstrates advantages in both the number of refinements and the overhead. This is attributed to its targeted counterexample generation and adaptive space refinement during the repair process, so that each sub-space can receive more surgical correction.

Remark 2: ProRepair outperforms current provable methods in handling region-wise tasks, achieving notable improvements in repair capability and computational efficiency.

6.3 How does the number of properties impact?

In this section, we investigate how all repair methods perform with a different number of desired properties (i.e., $|\mathcal{P}|$).

Backdoor. We first impose constraints on the number of properties in the backdoor repair scenario, ranging from 10 to 100. Results are presented in Fig.5 (CIFAR-100) and Fig.10 (full results, Appendix). For clarity, we omit the PSR metric, as all methods achieve 100% PSR when the repair is successful. However, as the number of properties increases, both APRNN and PRDNN encounter out-of-memory errors or infeasible LP issues (i.e., truncated curves in the figure), particularly on datasets with a large number of classes. In terms of accuracy, the results highlight that ProRepair exhibits exceptional stability in preserving the original knowledge of the model. In comparison, the results of APRNN and PRDNN are unstable, achieving satisfactory accuracy in a few scenarios. Moreover, we find that their generalization improves as the number of properties increases. This may be attributed to the fact that more properties force them to change the wrong behavior of more activation status. However, the number of activation status associated with the backdoor may far exceed the amount of data available to construct properties,

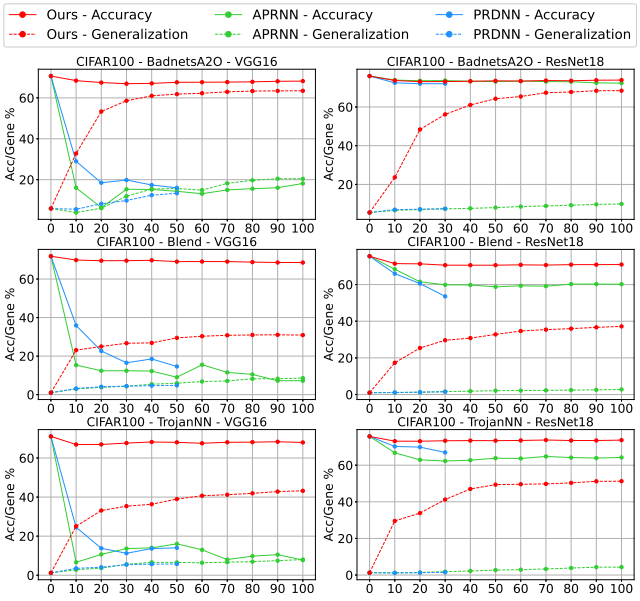


Figure 5: Results of backdoor repair on CIFAR-100 with different numbers of properties (x-axis).

which inherently limits their generalization. In contrast, ProRepair consistently demonstrates significantly higher generalization across all settings. Even with limited properties, it continues to exhibit considerable generalization.

For SEAM and I-REPAIR, we both vary the number of properties and additional training data. Specifically, we provide them with 1%, 2%, and 5% of training data. As shown in Fig. 11 (Appendix), both I-REPAIR and our method show an acceptable decline in accuracy. However, I-REPAIR struggles with more covert attack strategies, as neither more properties nor clean data improve its PSR and

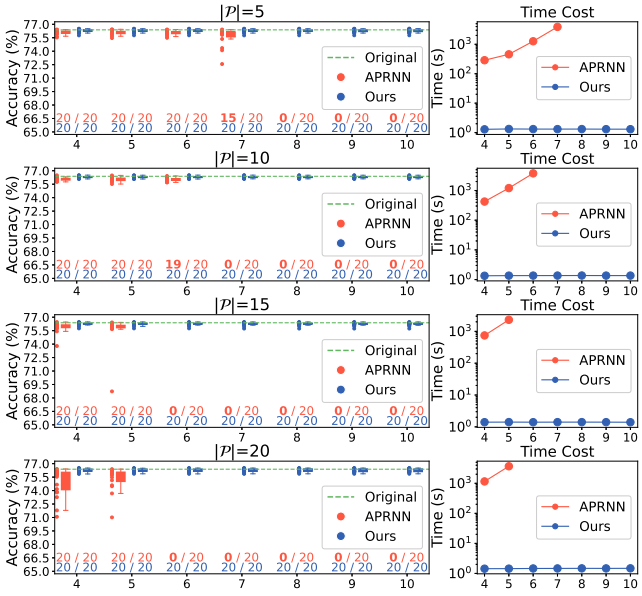


Figure 6: Results of adversarial attack repair with multiple properties on CIFAR-10 (x-axis: space dimensionality).

generalization under the Blend attack. On the other hand, SEAM's performance generally improves with more clean data. Overall, our method does not rely on additional clean data and outperforms both I-REPAIR and SEAM in most configurations.

Corruption. We conduct experiments on corruption repair with varying numbers of properties. The results of REASSURE are omitted as it shows 0% accuracy drop and generalization. Overall, Fig. 12 (Appendix) shows that ProRepair remains effective, maintaining stable accuracy and consistently achieving best generalization.

Adversarial Attack. We conduct an evaluation on adversarial attack repair with a single property in Sec. 6.2.3, while in practice a repairer may require the network to exhibit robustness across multiple local spaces. So we increase the number of properties involved in the repair, the results are shown in Fig. 6 and Fig. 13 (Appendix). As we can see, APRNN's performance gradually worsens as the number of properties increases. Specifically, when handling multiple local spaces, it struggles to scale beyond 5 (11, resp.) dimensions for CIFAR-10 (MNIST), whereas ProRepair shows negligible accuracy loss and remarkable efficiency.

> **Remark 3:** ProRepair continues to outperform baselines with various numbers of desired properties.

6.4 Ablation Study

The Validity of Proxy Box Synthesis. We first investigate the impact of the box synthesis algorithm, which aims to generate proxy boxed to characterize the entire preimages. Here we consider a gradient-based approach, where we optimize a single point $\boldsymbol{h}$ to represent the whole preimage based on its gradients w.r.t. the objective in (3), i.e., $\sum_{\psi \in \phi^{cons}} -\min\left(\boldsymbol{c}_\psi^\top \cdot f_c(\boldsymbol{h}) + d_\psi, 0\right)$. We conduct experiments on corruption and backdoor repair. The results are

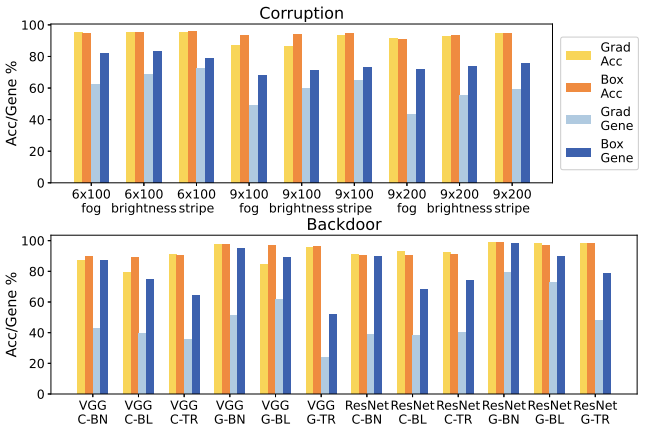


Figure 7: Comparison of proxy box generate algorithm with gradient-based method. C, G, BN, BL and TR denote CIFAR10, GTSRB, BadnetsA2O, Blend and TrojanNN.

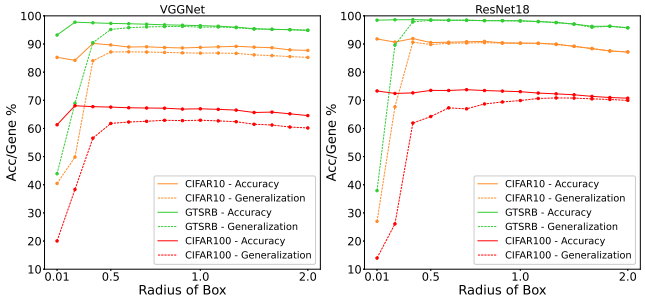


Figure 8: Impact of box radius on repair results.

shown in Fig. 7. Overall, both two methods maintain high accuracy, as they preserve the preimage of the feature space. However, the box synthesis algorithm shows significantly better generalization across all scenarios. Here are two main reasons for this improvement. First, it analyzes the model's behavior across the entire box during each update, preventing certain dimensions from falling into local optima and halting updates. Second, the center $\boldsymbol{h}_\phi^*$ returned by the box synthesis method promotes better generalization because the surrounding neighborhood (the entire box) stays within the preimage, which the gradient-based method cannot ensure.

Proxy Box Radius. We study the impact of the proxy box radius r, the results are shown in Fig. 8. As the radius of the box increases, we observe that the generalization of the repaired model improves consistently when the radius is relatively small. This is because a smaller radius may not be sufficient to jump out of local minima, which further supports the analysis in Section 5.1. Generally, the performance of ProRepair remains robust at a moderate radius. However, when the radius becomes excessively large (e.g., 2.0), the results show a slight decline, since the larger box increases the relaxation error in the box synthesis process.

The Validity of the Refine Score. We further evaluate the effectiveness of the *Refine Score* in region-wise repair, comparing it against a baseline metric that selects refinement dimensions solely

by the magnitude of the input range. On the global safety property repair task, the baseline repairs 34 out of 36 models with 357.7 steps and 7.5 seconds on average. In comparison, the *Refine Score* repairs all 36 models with only 150.0 steps and 4.4 seconds, demonstrating clear superiority in both efficiency and effectiveness.

6.5 Scalability

6.5.1 Other activation functions. To evaluate the scalability of all repair methods, we conducted a set of backdoor repair experiments under the BadnetsA2O attack, where we train NNs with various activation functions. Specifically, we consider Leaky ReLU, GeLU, and Tanh. The former is a piecewise linear function, while the latter two are more complex non-linear functions. PRDNN and APRNN are excluded as their implementation only supports ReLU activation at this model scale. As shown in Tab. 7, I-REPAIR exhibits unstable results across different datasets, while SEAM fails to maintain accuracy. In contrast, our method demonstrates impressive robustness when handling different activation functions.

6.5.2 High dimensional region. In Sec. 6.2.3, we assess APRNN and ProRepair on repairing relatively low-dimensional input spaces. Here we extend the evaluation to higher-dimensional regions, where APRNN faces significant challenges due to its reliance on enumerating the vertices of the input space. For each dimensionality, we perform 50 repair tasks on CIFAR-10 and record the number of successes along with the average accuracy after repair. Tab. 8 shows that ProRepair maintains a high success rate and preserves model performance even in spaces with dimensions up to 18 times higher than those that APRNN can handle (up to 9 for CIFAR-10).

To further explore the scalability of ProRepair, we conduct an additional experiment where all input dimensions are perturbed. Specifically, on the MNIST dataset, we construct a 784-dimensional space by allowing all pixels to be perturbed within a radius of 10^{-3}. This setting is challenging due to increased approximation errors in such high-dimensional regions, which hinder both counterexample generation and verification. Nevertheless, we run 50 repair tasks under this setting and observe that ProRepair successfully repairs 49 of them. In contrast, existing baselines such as APRNN are hardly feasible under these settings, as they rely on explicit vertex enumeration and scale only to 10–20 dimensions.

Remark 4: ProRepair greatly benefits from the box synthesis method and the refine score; it is robust to the proxy box radius and can scale to various activation functions and higher-dimensional spaces (18-fold improvement over SOTA).

7 Discussion

Limitations. ProRepair integrates a sound verifier to provide guarantees for region-wise repair and exhibits significantly improved scalability compared to prior provable methods. However, its scalability is fundamentally limited by the verifier itself, as it requires multiple invocations for verification. For example, when repairing high-dimensional spaces that are challenging for the existing verifier, ProRepair may engage in excessive property refinement but still fail to pass verification (few cases in Tab. 8). To further explore this issue, we incorporate different linear relaxation method based verifiers into ProRepair for detailed analysis. Tab. 9 (Appendix) shows that integrating stronger verifier significantly enhances it's capability to handle higher-dimensional spaces. We believe that using advanced verifiers will help ProRepair scale further.

Future work. While we apply ProRepair to four different repair tasks, additional types of failures, such as fairness issue, remain to be explored. Moreover, this work limits the input space for the desired properties to be a bounded box. It is worth exploring how ProRepair can be extended to handle properties defined over other types of input specifications such as ℓ_2-balls. Incorporating verifiers specialized for such regions (e.g., SDP-CROWN [8]) may yield tighter bounds and directly benefit the repair process. On the other hand, how to refine such regions becomes a core challenge, as traditional space partitioning does not directly apply. One possible approach is to iteratively add linear constraints through the center to partition the ball. We leave these extensions to future work to further enhance ProRepair's versatility.

Table 7: Repair with different activation functions.

Activation Function	Method	GTSRB PSR	GTSRB Acc	GTSRB Gene	GTSRB T	CIFAR-10 PSR	CIFAR-10 Acc	CIFAR-10 Gene	CIFAR-10 T
Leaky ReLU	Original	0	97.32	7.63	-	0	91.87	12.79	-
	SEAM	96	64.12	50.32	1.6	58	71.80	47.36	2.7
	IREPAIR	**100**	96.84	67.09	1.3	36	87.59	27.01	6.0
	Ours	**100**	**97.48**	**92.24**	**0.7**	**100**	**90.00**	**88.82**	**0.7**
GeLU	Original	0	97.50	7.69	-	0	90.83	13.21	-
	SEAM	92	53.81	49.11	1.8	56	69.36	52.34	**0.5**
	IREPAIR	**100**	92.89	75.08	1.5	22	83.67	19.69	6.9
	Ours	**100**	**96.07**	**92.16**	**1.1**	**100**	**88.96**	**74.02**	1.3
Tanh	Original	0	97.35	7.72	-	0	89.32	13.04	-
	SEAM	92	61.66	54.29	0.7	56	54.58	36.13	**0.5**
	IREPAIR	**100**	**97.51**	66.41	**0.4**	**100**	82.90	72.64	2.2
	Ours	**100**	95.95	**95.58**	0.8	**100**	**85.46**	**82.28**	1.1

Table 8: Results of repair with high dimensional region.

Dim	#Success	Accuracy	T	Dim	#Success	Accuracy	T
20	50 / 50	76.270±0.039	1.3	40	50 / 50	76.265±0.039	1.3
60	49 / 50	76.275±0.035	1.3	80	49 / 50	76.272±0.039	1.7
100	48 / 50	76.269±0.040	1.5	120	48 / 50	76.271±0.038	1.8
140	41 / 50	76.265±0.041	0.9	160	41 / 50	76.250±0.046	2.1

8 Conclusion

In this paper, we propose ProRepair, a provable neural network repair framework. The core idea is to synthesize small proxy boxes to effectively characterize the preimages and further guide the repair. ProRepair further employs an adaptive approach that includes counterexample generation and property refinement to address more practical tasks. Thorough evaluation across diverse repair tasks demonstrate that ProRepair exhibits remarkable effectiveness, efficiency and scalability compared to existing repair methods.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (U21B2001), the Key R&D Program of Zhejiang

(2022C01018), and the CCF-Huawei Populus Grove Fund (CCF-HuaweiFM2024003).

References

[1] Mauro Barni, Kassem Kallas, and Benedetta Tondi. 2019. A new backdoor attack in cnns by training set corruption without label poisoning. In *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE, 101–105.

[2] Ben Batten, Panagiotis Kouvaros, Alessio Lomuscio, and Yang Zheng. 2021. Efficient neural network verification via layer-based semidefinite relaxations and linear cuts. In *International Joint Conference on Artificial Intelligence*. 2184–2190.

[3] Stephen P Boyd and Lieven Vandenberghe. 2004. *Convex optimization*. Cambridge university press.

[4] Jialuo Chen, Jingyi Wang, Youcheng Sun, Peng Cheng, and Jiming Chen. 2024. Isolation-Based Debugging for Neural Networks. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 338–349.

[5] 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).

[6] Zuohui Chen, Jun Zhou, Youcheng Sun, Jingyi Wang, Qi Xuan, and Xiaoniu Yang. 2024. Interpretability Based Neural Network Repair. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 908–919.

[7] Zhiming Chi, Jianan Ma, Pengfei Yang, Cheng-Chao Huang, Renjue Li, Jingyi Wang, Xiaowe Huang, and Lijun Zhang. 2025. Patch Synthesis for Property Repair of Deep Neural Networks. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE, 1191–1203.

[8] Hong-Ming Chiu, Hao Chen, Huan Zhang, and Richard Y Zhang. [n. d.]. SDP-CROWN: Efficient Bound Propagation for Neural Network Verification with Tightness of Semidefinite Programming. In *Forty-second International Conference on Machine Learning*.

[9] Andrew C Cullen, Paul Montague, Shijie Liu, Sarah M Erfani, and Benjamin IP Rubinstein. 2024. It's Simplex! Disaggregating Measures to Improve Certified Robustness. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2886–2900.

[10] 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.

[11] Ruediger Ehlers. 2017. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 269–286.

[12] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. 2018. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1625–1634.

[13] 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.

[14] 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.

[15] Xueluan Gong, Yanjiao Chen, Wang Yang, Qian Wang, Yuzhe Gu, Huayang Huang, and Chao Shen. 2023. Redeem myself: Purifying backdoors in deep learning models using self attention distillation. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 755–772.

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

[17] 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.

[18] Yong Guo, David Stutz, and Bernt Schiele. 2022. Improving robustness by enhancing weak subnets. In *European Conference on Computer Vision*. Springer, 320–338.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

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

[21] Dan Hendrycks, Norman Mu, Ekin D Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. 2019. Augmix: A simple data processing method to improve robustness and uncertainty. *arXiv preprint arXiv:1912.02781* (2019).

[22] 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.

[23] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 2017. Safety verification of deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*. Springer, 3–29.

[24] 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).

[25] 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.

[26] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *CAV 2017*. Springer, Heidelberg, Germany, 97–117.

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

[28] Suhas Kotha, Christopher Brix, J Zico Kolter, Krishnamurthy Dvijotham, and Huan Zhang. 2023. Provably bounding neural network preimages. *Advances in Neural Information Processing Systems* 36 (2023), 80270–80290.

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

[30] Matan Levi and Aryeh Kontorovich. 2024. Splitting the difference on adversarial training. In *33rd USENIX Security Symposium (USENIX Security 24)*. 3639–3656.

[31] Zhen Liang, Taoran Wu, Changyuan Zhao, Wanwei Liu, Bai Xue, Wenjing Yang, Ji Wang, and Wanrong Huang. 2025. BIRDNN: Behavior-Imitation Based Repair for Deep Neural Networks. *Neural Networks* 183 (2025), 106949.

[32] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. 2019. Abs: Scanning neural networks for back-doors by artificial brain stimulation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1265–1282.

[33] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2018. Trojaning attack on neural networks. In *25th Annual Network And Distributed System Security Symposium (NDSS 2018)*.

[34] 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 *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–13.

[35] Guido Manfredi and Yannick Jestin. 2016. An introduction to ACAS Xu and the challenges ahead. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. IEEE, 1–9.

[36] Mike Marston and Gabe Baca. 2015. *ACAS-Xu initial self-separation flight tests*. Technical Report.

[37] Kyle Matoba and François Fleuret. 2020. Exact preimages of neural network aircraft collision avoidance systems. In *Proceedings of the Machine Learning for Engineering Modeling, Simulation, and Design Workshop at Neural Information Processing Systems*. 1–9.

[38] Xiaoxing Mo, Yechao Zhang, Leo Yu Zhang, Wei Luo, Nan Sun, Shengshan Hu, Shang Gao, and Yang Xiang. 2024. Robust backdoor detection for deep learning via topological evolution dynamics. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 171–171.

[39] Norman Mu and Justin Gilmer. 2019. Mnist-c: A robustness benchmark for computer vision. *arXiv preprint arXiv:1906.02337* (2019).

[40] 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.

[41] 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).

[42] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[43] Gagandeep Singh. 2019. ETH Robustness Analyzer for Neural Networks (ERAN). https://github.com/eth-sri/eran

[44] 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).

[45] 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.

[46] Jeongju Sohn, Sungmin Kang, and Shin Yoo. 2023. Arachne: Search-based repair of deep neural networks. *ACM Transactions on Software Engineering and Methodology* 32, 4 (2023), 1–26.

[47] Matthew Sotoudeh and Aditya V Thakur. 2021. Provable repair of deep neural networks. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 588–603.

[48] 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.
[49] 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.
[50] Zhe Tao, Stephanie Nawas, Jacqueline Mitchell, and Aditya V Thakur. 2023. Architecture-preserving provable repair of deep neural networks. *Proceedings of the ACM on Programming Languages* 7, PLDI (2023), 443–467.
[51] Florian Tramer and Dan Boneh. 2019. Adversarial training and robustness for multiple perturbations. *Advances in neural information processing systems* 32 (2019).
[52] Muhammad Usman, Divya Gopinath, Youcheng Sun, Yannic Noller, and Corina S Păsăreanu. 2021. NN repair: constraint-based repair of neural network classifiers. In *Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I 33*. Springer, 3–25.
[53] Jie Wan, Jianhao Fu, Lijin Wang, and Ziqi Yang. 2023. Bounceattack: A query-efficient decision-based adversarial attack by bouncing into the wild. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 68–68.
[54] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. 2019. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE symposium on security and privacy (SP)*. IEEE, 707–723.
[55] Hang Wang, Zhen Xiang, David J Miller, and George Kesidis. 2024. Mm-bd: Post-training detection of backdoor attacks with arbitrary backdoor pattern types using a maximum margin statistic. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1994–2012.
[56] Haotao Wang, Chaowei Xiao, Jean Kossaifi, Zhiding Yu, Anima Anandkumar, and Zhangyang Wang. 2021. Augmax: Adversarial composition of random augmentations for robust training. *Advances in neural information processing systems* 34 (2021), 237–250.
[57] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. 2021. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in Neural Information Processing Systems* 34 (2021), 29909–29921.
[58] Lily Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. 2018. Towards fast computation of certified robustness for relu networks. In *International Conference on Machine Learning*. PMLR, 5276–5285.
[59] Chong Xiang, Arjun Nitin Bhagoji, Vikash Sehwag, and Prateek Mittal. 2021. {PatchGuard}: A provably robust defense against adversarial patches via small receptive fields and masking. In *30th USENIX Security Symposium (USENIX Security 21)*. 2237–2254.
[60] Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, Alan L Yuille, and Quoc V Le. 2020. Adversarial examples improve image recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 819–828.
[61] 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.
[62] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. 2020. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. *arXiv preprint arXiv:2011.13824* (2020).
[63] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A Gunter, and Bo Li. 2021. Detecting ai trojans using meta neural analysis. In *2021 IEEE Symposium on Security and Privacy (SP)*. 103–120.
[64] 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.
[65] Kaiyuan Zhang, Siyuan Cheng, Guangyu Shen, Guanhong Tao, Shengwei An, Anuran Makur, Shiqing Ma, and Xiangyu Zhang. 2024. Exploring the Orthogonality and Linearity of Backdoor Attacks. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 225–225.
[66] Xinyu Zhang, Hanbin Hong, Yuan Hong, Peng Huang, Binghui Wang, Zhongjie Ba, and Kui Ren. 2024. Text-crs: A generalized certified robustness framework against textual adversarial attacks. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2920–2938.
[67] Xiyue Zhang, Benjie Wang, and Marta Kwiatkowska. 2024. Provable preimage under-approximation for neural networks. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 3–23.
[68] 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.
[69] Runkai Zheng, Rongjun Tang, Jianze Li, and Li Liu. 2022. Pre-activation distributions expose backdoor neurons. *Advances in Neural Information Processing Systems* 35 (2022), 18667–18680.
[70] Hong Zhu, Shengzhi Zhang, and Kai Chen. 2023. Ai-guardian: Defeating adversarial attacks using backdoors. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 701–718.
[71] Rui Zhu, Di Tang, Siyuan Tang, XiaoFeng Wang, and Haixu Tang. 2023. Selective amnesia: On efficient, high-fidelity and blind suppression of backdoor effects in trojaned machine learning models. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1–19.

A Proofs of Theorems

A.1 Proof of Theorem 1

Proof. We first prove the inequality chain in Eq. (6). Consider two cases:

(i) When $\tilde{f}_{\mathrm{e}}(\boldsymbol{x}_\phi) \in \mathcal{B}_\phi$, the left inequality holds trivially since $\mathcal{L}_{\mathrm{Proj}} = 0$ and $\|\boldsymbol{h}^*_\phi - \Pi_{\mathcal{B}_\phi}\tilde{f}_{\mathrm{e}}(\boldsymbol{x}_\phi)\|_p = \|\boldsymbol{h}^*_\phi - \tilde{f}_{\mathrm{e}}(\boldsymbol{x}_\phi)\|_p = \mathcal{L}$. The right inequality becomes: $\mathcal{L} \le r \cdot \mathrm{s}^{1/p}$, which holds by $\|\tilde{f}_{\mathrm{e}}(\boldsymbol{x}_\phi) - \boldsymbol{h}^*_\phi\|_p \le \mathrm{s}^{1/p}\|\tilde{f}_{\mathrm{e}}(\boldsymbol{x}_\phi) - \boldsymbol{h}^*_\phi\|_\infty \le \mathrm{s}^{1/p} \cdot r$.

(ii) If $\tilde{f}_{\mathrm{e}}(\boldsymbol{x}_\phi) \notin \mathcal{B}_\phi$, the left inequality follows from the definition of projection operator [3] and the geometric nature of the proxy box:

- $\mathcal{L}_{\mathrm{Proj}} = \min_{\boldsymbol{h}\in\mathcal{B}_\phi}\|\tilde{f}_{\mathrm{e}}(\boldsymbol{x}_\phi) - \boldsymbol{h}\|_p \le \|\tilde{f}_{\mathrm{e}}(\boldsymbol{x}_\phi) - \boldsymbol{h}^*_\phi\|_p = \mathcal{L}$;
- Let $\boldsymbol{v} = \tilde{f}_{\mathrm{e}}(\boldsymbol{x}_\phi) - \boldsymbol{h}^*_\phi$ and $\tilde{\boldsymbol{v}} = \Pi_{\mathcal{B}_\phi}\tilde{f}_{\mathrm{e}}(\boldsymbol{x}_\phi) - \boldsymbol{h}^*_\phi$. Since the box projection performs coordinate-wise truncation, we have: $|\tilde{v}_i| = \min(|v_i|, r) \le |v_i|$. Therefore, for any $p \ge 1$:

$$\|\tilde{\boldsymbol{v}}\|_p = \left(\sum_{i=1}^{\mathrm{s}} |\tilde{v}_i|^p\right)^{1/p} \le \left(\sum_{i=1}^{\mathrm{s}} |v_i|^p\right)^{1/p} = \|\boldsymbol{v}\|_p$$

which proves $\|\Pi_{\mathcal{B}_\phi}\tilde{f}_{\mathrm{e}}(\boldsymbol{x}_\phi) - \boldsymbol{h}^*_\phi\|_p \le \mathcal{L}$.

To sum up, we have $\max\left(\|\boldsymbol{h}^*_\phi - \Pi_{\mathcal{B}_\phi}\tilde{f}_{\mathrm{e}}(\boldsymbol{x}_\phi)\|_p\,,\ \mathcal{L}_{\mathrm{Proj}}\right) \le \mathcal{L}$. For the right inequality, by the Minkowski inequality we have $\mathcal{L} \le \mathcal{L}_{\mathrm{Proj}} + \|\Pi_{\mathcal{B}_\phi}\tilde{f}_{\mathrm{e}}(\boldsymbol{x}_\phi) - \boldsymbol{h}^*_\phi\|_p$ where $\|\Pi_{\mathcal{B}_\phi}\tilde{f}_{\mathrm{e}}(\boldsymbol{x}_\phi) - \boldsymbol{h}^*_\phi\|_p \le r \cdot \mathrm{s}^{1/p}$ since $\tilde{f}_{\mathrm{e}}(\boldsymbol{x}_\phi)$ is on the surface of $\mathcal{B}_\phi$.

Combining cases (i) and (ii), We conclude that Eq. (6) holds.

The repair guarantee follows as $\mathcal{L} \le r$ implies $\|\tilde{f}_{\mathrm{e}}(\boldsymbol{x}_\phi) - \boldsymbol{h}^*_\phi\|_\infty \le r$ by ℓ_p-norm monotonicity, placing $\tilde{f}_{\mathrm{e}}(\boldsymbol{x}_\phi)$ in $\mathcal{B}_\phi$. Since $\mathcal{B}_\phi \subset f_{\mathrm{c}}^{-1}(\mathbb{R}^{\mathrm{s}}, \phi^{out})$, we have $f_{\mathrm{c}}(\tilde{f}_{\mathrm{e}}(\boldsymbol{x}_\phi)) \in \phi^{out}$. □

A.2 Proof of Theorem 2

Proof. Algorithm 2 returns a repaired model $\tilde{f}$ only at line 26, i.e., only if the unsatisfied constraints set $\mathrm{Vio}(\phi)$ for all properties $\phi \in \mathcal{P}$ is empty. This condition implies: $lb_\psi \ge 0,\ \forall \phi \in \mathcal{P}, \psi \in \phi^{cons}$. From the linear bound definition:

$$lb_\psi = \min_{\boldsymbol{x}\in\phi^{in}}\left(\underline{\boldsymbol{w}}_\psi^\top \boldsymbol{x} + \underline{b}_\psi\right) \le \underline{\boldsymbol{w}}_\psi^\top \boldsymbol{x} + \underline{b}_\psi \le \boldsymbol{c}_\psi^\top \tilde{f}(\boldsymbol{x}) + d_\psi,\ \ \forall \boldsymbol{x} \in \phi^{in}$$

we conclude $\boldsymbol{c}_\psi^\top f(\boldsymbol{x}) + d_\psi \ge 0$ for all $\boldsymbol{x} \in \phi^{in}$, $\psi \in \phi^{cons}$ and $\phi \in \mathcal{P}$. Finally. we have $\forall \phi \in \mathcal{P} :\ \tilde{f} \models \phi$. □

B Experimental Details

Platform. We conducted all the experiments on a machine with Dual AMD EPYC 7763 64-Core Processor with 256 GB of memory and RTX-4090 running Ubuntu 20.04.

B.1 Details of Repair Tasks

B.1.1 Backdoor. For this task, we train VGGNet [42] and ResNet-18 [19] models on four datasets including CIFAR-10 [29], SVHN [41],

GTSRB [48] and CIFAR-100 [29]. We consider five common backdoor attacks, i.e., BadNets [17] using two attack strategies (All to One and All to All), TrojanNN[33], Blend [5] and SIG [1]. For each dataset, we inject the backdoor trigger into its test set $\mathcal{D}$ to obtain **the poisoned set** $\tilde{\mathcal{D}}$ and divide it into two disjoint sets: **the repair set** $\tilde{\mathcal{D}}_{\mathrm{r}}$ consisting of 1 000 data and **the generalization set** $\tilde{\mathcal{D}}_{\mathrm{g}}$ comprising the remaining data. We randomly select **a few buggy data** from **the repair set** to construct our desired property set $\mathcal{P} = \{\phi_1, \ldots, \phi_p\}$, i.e., each property ϕ corresponds to a buggy data $\boldsymbol{x}$, along with the output constraints $\bigwedge_{j=1}^{n} f(\boldsymbol{x})_{cl(\boldsymbol{x})} - f(\boldsymbol{x})_j \geq 0$, where $cl(\boldsymbol{x})$ is the correct label of $\boldsymbol{x}$. Finally, the repaired network is assessed on datasets $\mathcal{D}$ and $\tilde{\mathcal{D}}_{\mathrm{g}}$ to measure its *Acc* and *Gene*.

B.1.2 Corruption. The MNIST-C dataset includes corrupted images from the original MNIST test set. We consider all 15 corruption types, such as *fog*, *brightness*, *stripe*, etc. Each corruption set consists of 10,000 images, which we divide into a **repair set** (1 000) and a **generalization set** (9 000). From **the repair set**, we further randomly selected a subset of buggy data to construct **the desired property set** $\mathcal{P}$. We evaluate the repaired model on the MNIST test set and the generalization set to measure its *Acc* and *Gene*.

B.1.3 Adversarial Attack. In this task, we first randomly select several buggy data from the original training set and then construct the specification for each buggy data $\boldsymbol{x}$. Following APRNN, we perturb the first k non-zero pixels of the input to construct the desirable property for $\boldsymbol{x}$ as follows:

$$\phi^{in} = \{\boldsymbol{x}' \mid \forall p \in \mathrm{P}, |\boldsymbol{x}'_p - \boldsymbol{x}_p| \leq \mathrm{R},\ \forall q \in \mathrm{NP}, \boldsymbol{x}'_q = \boldsymbol{x}_q\}$$
$$\phi^{cons} = \{f(\boldsymbol{x})_{cl(\boldsymbol{x})} - f(\boldsymbol{x})_j \geq 0 \mid 1 \leq j \leq n\}$$

where P and NP represent the set of indices of the first k non-zero pixels and the remaining pixels, respectively. R is the perturbation radius. Specifically, the perturbation radii are set to 0.1 for MNIST, and 2/255 for both GTSRB and CIFAR-10.

B.1.4 Safety Property Violation. This task aims to correct the buggy model in the ACAS Xu benchmark so that it satisfies the predefined global safety property. ACAS Xu system [35] is an aircraft collision avoidance system designed for unmanned aircraft. The input of DNNs for this system consists of five variables describing the speed (v_{int} and v_{own}) and relative position (relative distance ρ and angles θ, ψ) of both the intruder and ownship, and the outputs are the scores of five advisories that can be given to the ownship: Clear-of-Conflict (COC), weak right (WR), strong right (SR), weak left (WL), or strong left (SL). These models are subject to a set of safety-critical properties [26]. Following prior repair frameworks [31, 47, 49], we consider Property-2:

- **Description:** If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will be minimal (ACAS Xu system chooses the lowest score as the best action).
- **Desired Property**:

$$\phi^{in} = \{(\rho, \theta, \psi, v_{own}, v_{int}) \mid \rho \geq 55947.691, v_{own} \geq 1145, v_{int} \leq 60\}$$
$$\phi^{cons} = \{f(\cdot)_{COC} \leq f(\cdot)_j \mid j \in \{WR, SR, SL, WL\}\}$$

B.2 Details of Models

Corruption. The models used for corruption repair are sourced from ERAN [43], a robustness platform for DNNs. As in APRNN, we refer to the networks by the number of layers and the width of hidden layers, e.g., a "6 × 100" network has 6 hidden layers with 100 neurons per layer.

Backdoor. We use SGD as the optimizer to train the backdoored model with learning rate 0.1, momentum 0.9 and batch size 128 for 50 epochs on SVHN and GTSRB, 100 epochs on CIFAR-10 and CIFAR-100.

Adversarial Attack. The 3x100 model for MNIST is sourced from ERAN [43]. Other models are trained using SGD with learning rate 0.1, momentum 0.9 and batch size 128 for 20 epochs on GTSRB and 100 epochs on CIFAR-10.

Safety Property Violation. As reported in [10], over 30 out of the 45 DNNs in the ACAS Xu system violate Property 2. We conduct a comprehensive evaluation of all these models.

B.3 Details of Backdoor Attacks

We implement various backdoor attack as follows:

- **Badnets** [17]: For *All to One attack*, the backdoor trigger is a 5 × 5 white square at the bottom right corner of images. In the *All to All attack*, the target labels for backdoored data with original labels y are set to $y_t = (y + 1) \mod n$. Here mod is short for "modulus". The poisoning rate is set to be 10%.
- **Blend** [5]: We randomly generate a trigger pattern $\boldsymbol{t}$ by sampling pixel values from a uniform distribution in [0, 255]. Then we obtain the backdoored data by attaching the trigger $\boldsymbol{t}$ to the sample $\boldsymbol{x}$ and setting the blended ratio to 0.2, i.e., $\tilde{\boldsymbol{x}} = 0.2 \cdot \boldsymbol{t} + 0.8 \cdot \boldsymbol{x}$. Given the target class, we randomly poison 10% of training samples.
- **TrojanNN** [33]: Following [65], we use a reversed watermark as the trigger pattern. The threshold for the mask used to attach the trigger is set to 0.3. Given the target label, we attach the trigger to 10% of training samples for poisoning.
- **SIG** [1]: Following [1], we overlay a sinusoidal signal over the inputs as the trigger. Given the target label, we attach the trigger to 80% of samples from the target class, i.e., the actual poisoning rate is $\frac{80}{n}$% for the entire dataset, where n is the number of classes.

B.4 Details of Setup in Repair Methods

Here we briefly introduce the setup of all methods, especially hyper-parameter settings. For fundamental hyper-parameters such as learning rate and number of epochs, we adhere to the settings in the original papers. For other parameters, we provide the following explanations:

- **lb & ub**: In PRDNN and APRNN, this parameter restricts the modifications to the model ($\pm \|\Delta\theta\|_\infty$). Following APRNN, we set it to ±3 for corruption repair, ±5 for global safety property repair, and ±10 for both adversarial attack and backdoor repairs.
- **Rows**: In PRDNN and APRNN, this parameter is designed for large networks (such as VGG and ResNet), as repairing all parameters in a layer requires an unacceptable amount of memory. We set it according to the original setting to 400-800 for VGG and 0-200 for ResNet, where 400-800 indicates the number of rows in the parameter matrix to be repaired for one layer.
- **Layers**: In PRDNN, APRNN and our method, this parameter involves how to select the layers for repair (i.e., how to partition the network f into f_e and f_c). For PRDNN and APRNN, we try

Table 9: Results of adversarial attack repair with various verifiers. auto-LiRPA is the most precise verifier, followed by CROWN-Forward, then IBP.

Dim→ Verifier↓	20 #Success	40 #Success	60 #Success	80 #Success	100 #Success	120 #Success	140 #Success	160 #Success
IBP	49 / 50	46 / 50	39 / 50	22 / 50	11 / 50	6 / 50	1 / 50	0 / 50
CROWN-Forward	**50 / 50**	**50 / 50**	48 / 50	48 / 50	**48 / 50**	45 / 50	40 / 50	34 / 50
auto-LiRPA	**50 / 50**	**50 / 50**	**49 / 50**	**49 / 50**	**48 / 50**	**48 / 50**	**41 / 50**	**41 / 50**

our best to select the best-performing layer. For our method, we set parameters uniformly based on the model size. For small networks, we include one activation layer in f_c, while for larger networks, we include two activation layers.

Additionally, we provide further details on the configuration of SEAM [71], as it is the latest state-of-the-art method for backdoor removal based on a certain amount of clean samples. It first utilizes a small set of clean samples $\mathcal{D}_u$ (with wrong labels) for unlearning, and then uses a relatively larger set of clean samples $\mathcal{D}_r$ (labeled correctly) for recovery. In our scenarios, we made additional modifications to SEAM to enhance its performance.

- **Unlearning**: We provide the buggy data that used to construct property set $\mathcal{P}$ for SEAM to facilitate the unlearning process. Compared to using clean samples with incorrect labels, this operation allows for a more effective unlearning of the backdoor.
- **Recover**: During the recovery phase, we provide SEAM with additional clean samples, enabling it to utilize both the buggy data set (used in unlearning step) and the extra clean samples for recovery.

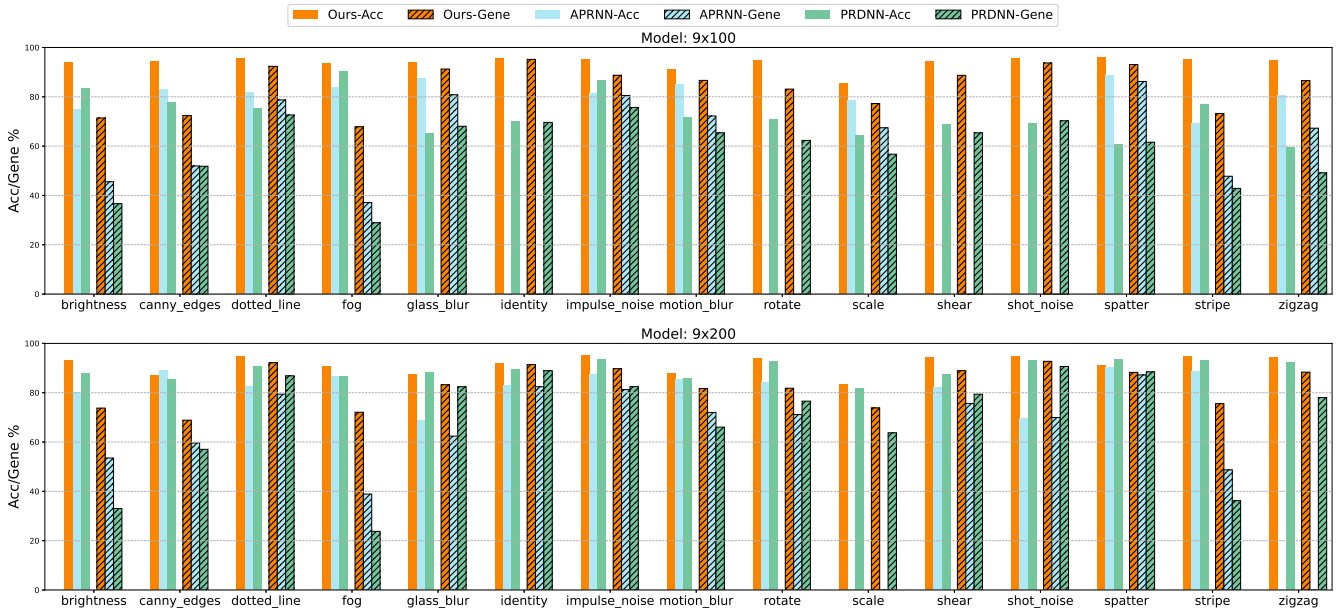


Figure 9: Results of point-wise corruption repair on the 9×100 and 9x200 models under different corruptions. We omit the bar if the corresponding repair is infeasible (e.g., APRNN on the 9×200 model under the scale corruption).

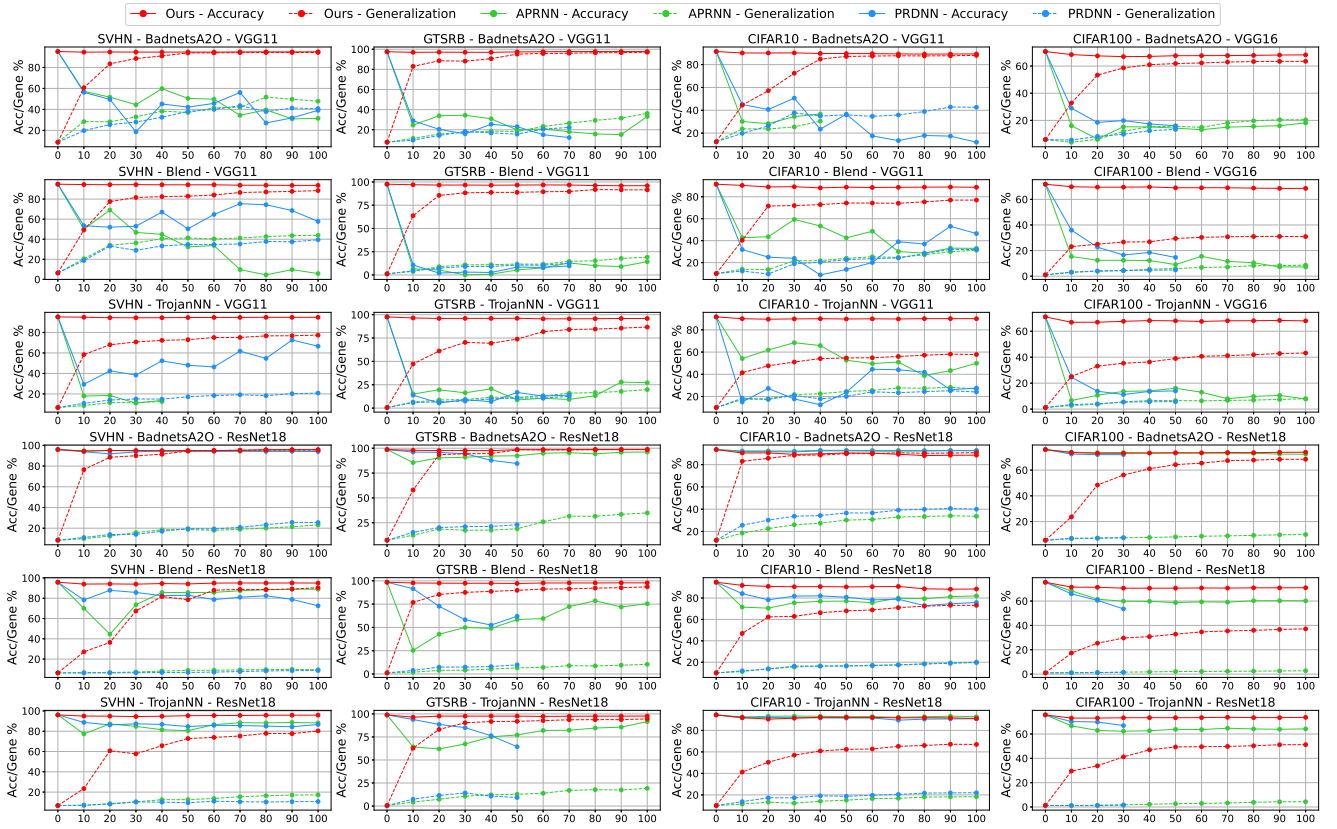
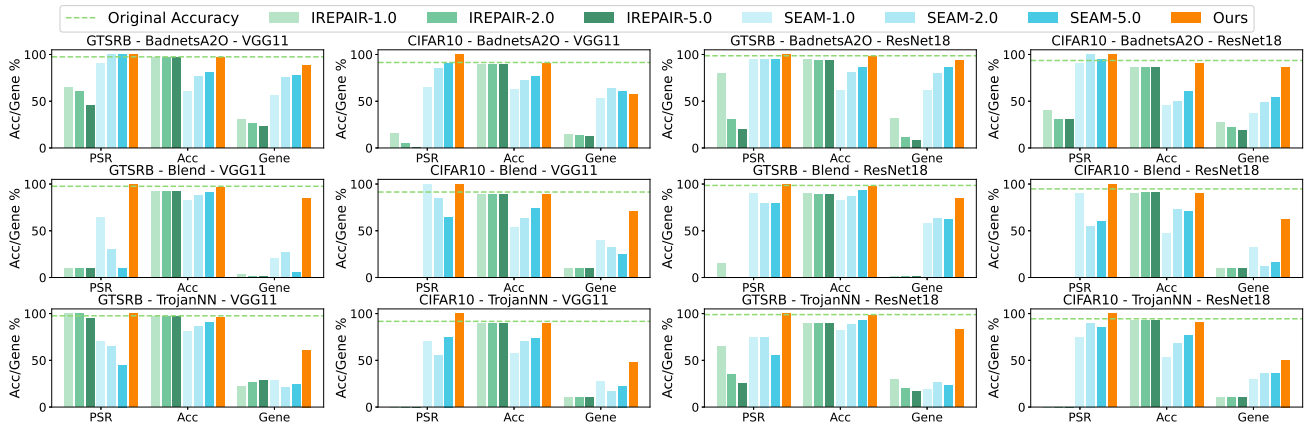
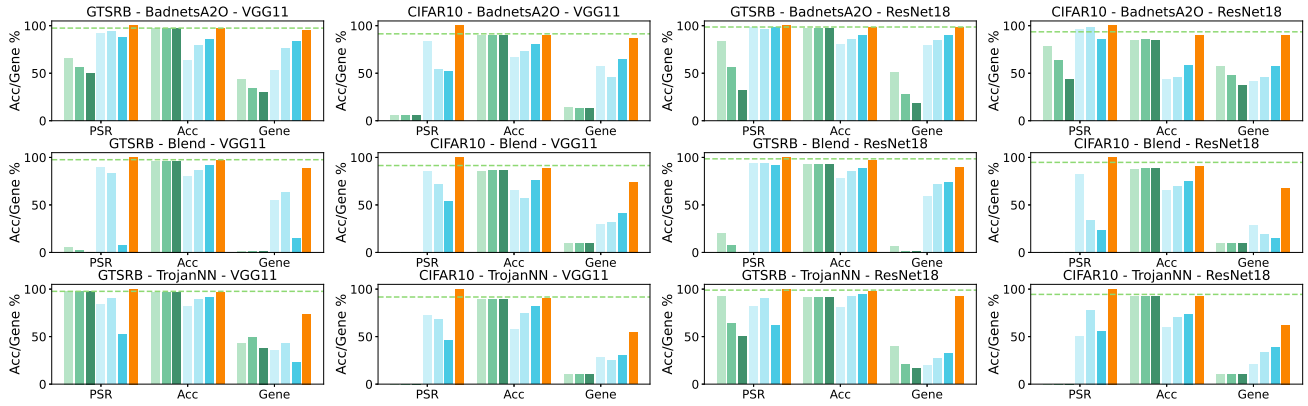


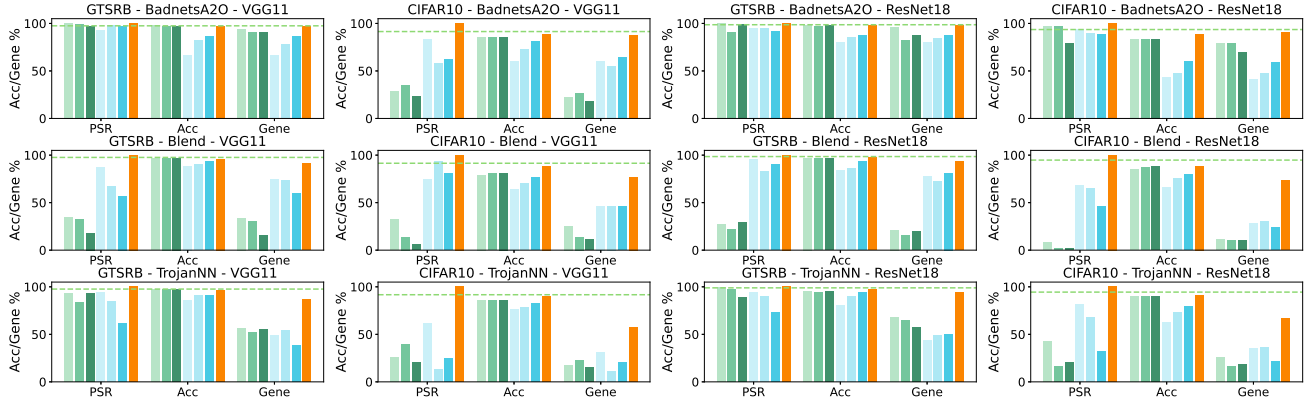
Figure 10: Full results of backdoor repair with different numbers of desired properties (x-axis).



(a) $|\mathcal{P}| = 20$



(b) $|\mathcal{P}| = 50$



(c) $|\mathcal{P}| = 100$

Figure 11: Comparison of ProRepair with SEAM-X and IREPAIR-X with different number of properties. X denotes the percentage of additional data accessible, e.g., SEAM-1.0 indicates that 500 additional clean data is available for CIFAR-10.

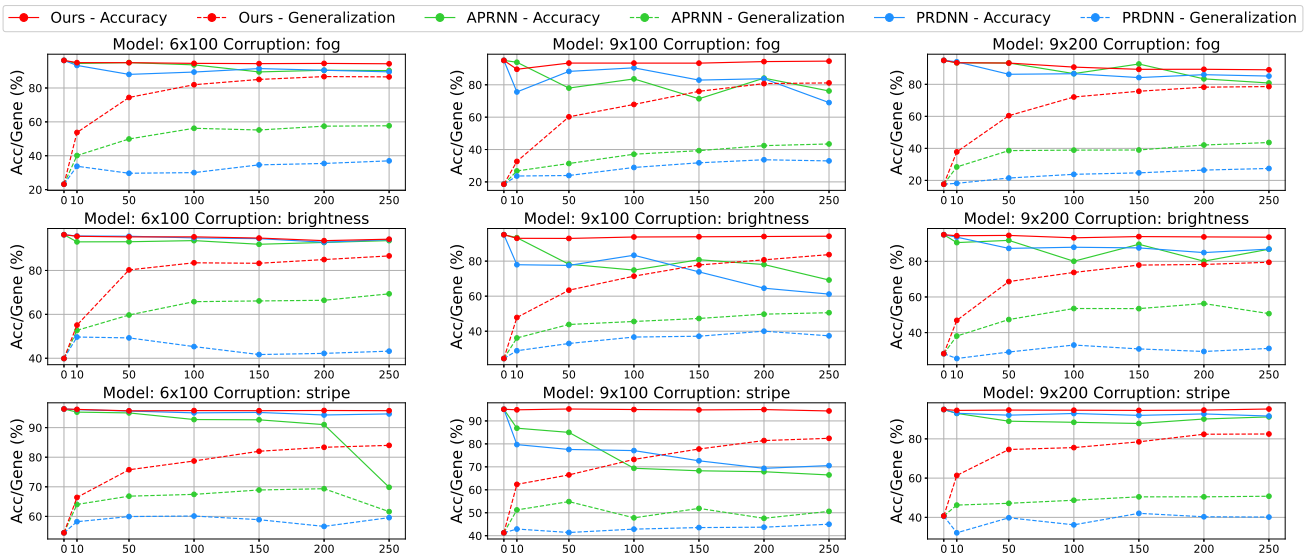


Figure 12: Results of corruption repair with different numbers of desirable properties (x-axis).

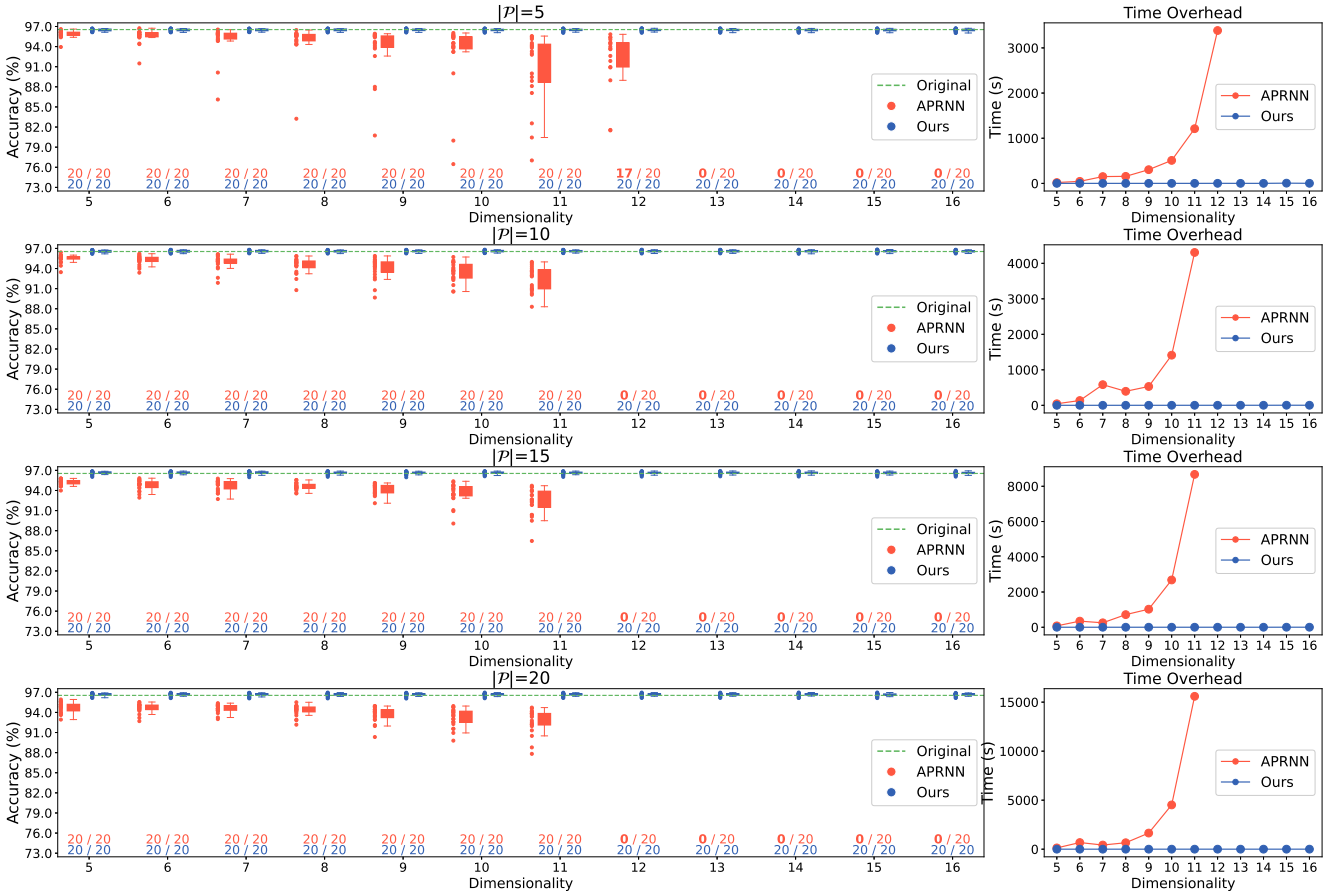


Figure 13: Results of adversarial attack repair with multiple properties on MNIST dataset.