Architecture-Preserving Provable Repair of Deep Neural Networks

ZHE TAO, University of California, Davis, U.S.A.
STEPHANIE NAWAS, University of California, Davis, U.S.A.
JACQUELINE MITCHELL, University of California, Davis, U.S.A.
ADITYA V. THAKUR, University of California, Davis, U.S.A.

Deep neural networks (DNNs) are becoming increasingly important components of software, and are considered the state-of-the-art solution for a number of problems, such as image recognition. However, DNNs are far from infallible, and incorrect behavior of DNNs can have disastrous real-world consequences. This paper addresses the problem of architecture-preserving V-polytope provable repair of DNNs. A V-polytope defines a convex bounded polytope using its vertex representation. V-polytope provable repair guarantees that the repaired DNN satisfies the given specification on the infinite set of points in the given V-polytope. An architecture-preserving repair only modifies the parameters of the DNN, without modifying its architecture. The repair has the flexibility to modify multiple layers of the DNN, and runs in polynomial time. It supports DNNs with activation functions that have some linear pieces, as well as fully-connected, convolutional, pooling and residual layers. To the best our knowledge, this is the first provable repair approach that has all of these features. We implement our approach in a tool called APRNN. Using MNIST, ImageNet, and ACAS Xu DNNs, we show that it has better efficiency, scalability, and generalization compared to PRDNN and REASSURE, prior provable repair methods that are not architecture preserving.

CCS Concepts: • **Computing methodologies → Neural networks**; • **Theory of computation → Linear programming**; • **Software and its engineering → Software post-development issues**.

Additional Key Words and Phrases: Deep Neural Networks, Repair, Bug fixing, Synthesis

ACM Reference Format:
Zhe Tao, Stephanie Nawas, Jacqueline Mitchell, and Aditya V. Thakur. 2023. Architecture-Preserving Provable Repair of Deep Neural Networks. *Proc. ACM Program. Lang.* 7, PLDI, Article 124 (June 2023), 25 pages. https://doi.org/10.1145/3591238

1 INTRODUCTION

Deep Neural Networks (DNNs) [Goodfellow et al. 2016], which learn by generalizing from a finite set of examples, are increasingly becoming critical components of software. They have emerged as the state-of-the-art approach for solving problems such as image recognition [Dosovitskiy et al. 2021; Iandola et al. 2016; Krizhevsky et al. 2012] and natural-language processing [Devlin et al. 2019; Liu et al. 2019; Ouyang et al. 2022]. They are being applied in diverse problem domains such as scientific computing [Stevens et al. 2020], medicine [Goodfellow et al. 2016; LR et al. 2021], and economics [Maliar et al. 2021]. Importantly, they are being deployed in safety-critical applications

Authors' addresses: Zhe Tao, Computer Science, University of California, Davis, Davis, California, 95616, U.S.A., zhetao@ucdavis.edu; Stephanie Nawas, Computer Science, University of California, Davis, Davis, California, 95616, U.S.A., snawas@ucdavis.edu; Jacqueline Mitchell, Computer Science, University of California, Davis, Davis, California, 95616, U.S.A., jlmitchell@ucdavis.edu; Aditya V. Thakur, Computer Science, University of California, Davis, Davis, California, 95616, U.S.A., avthakur@ucdavis.edu.

2475-1421/2023/6-ART124
https://doi.org/10.1145/3591238

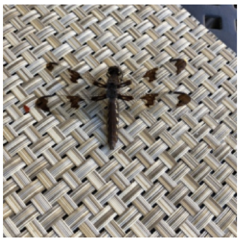


Fig. 1. Dragonfly misclassified as manhole cover.

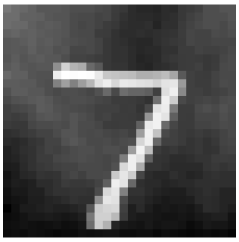


Fig. 2. Digit "7" misclassified as digit "5".

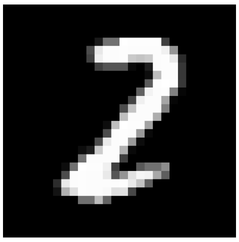


Fig. 3. Correctly classified digit "2".

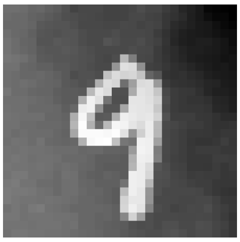


Fig. 4. Digit "9" misclassified as digit "8".

such as aircraft controllers [Julian et al. 2018]. However, DNNs are far from infallible, and incorrect behavior of DNNs can have disastrous real-world consequences. Thus, there has been extensive research on formal methods for the analysis and verification of DNNs [Ferrari et al. 2022; Katz et al. 2017; Lahav and Katz 2021; Müller et al. 2022; Singh et al. 2018; Xu et al. 2021]. In contrast, this paper addresses the next step of this process: what to do once faulty behavior is identified.

Consider the scenario in which we are given a (finite) set of inputs for which a DNN gives an incorrect result. For instance, Figure 1 shows an image of dragonfly that an ImageNet DNN misclassifies as manhole cover; Figure 2 shows an image of handwritten "7" that is misclassified as "5". *Pointwise repair of DNNs* takes as input a finite set of inputs along with the specification of their corresponding behavior, and returns a repaired DNN satisfying this repair specification. A DNN repair approach should satisfy the following properties:

- **P1** *Low repair drawdown.* Repairs to the DNN should not cause significant *drawdown*, which is when the DNN "forgets" its previous correct behavior on other inputs. For example, suppose a DNN correctly classifies the image in Figure 3 as a "2". Repairing the DNN to correctly classify the image in Figure 2 should not result in the incorrect classification of the image in Figure 3.
- **P2** *High repair generalization.* Repairing the DNN on one set of inputs should result in the repair of a similar set of inputs. For example, repairing the DNN for the fog-corrupted image in Figure 2 should also fix the classification for the fog-corrupted image in Figure 4.
- **P3** *Efficacy.* The repair should *guarantee* that the changes to the DNN result in correct outputs for the inputs according to the repair specification. For example, a repair of a faulty input to an aircraft collision avoidance DNN will guarantee that the output of the repaired DNN satisfies the specified safety properties.
- **P4** *Efficient.* The repair process should be efficient and scale to real-world networks.

One approach to pointwise repair could be to *retrain* the DNN by adding the faulty inputs with their corrected output to the original training set. Unfortunately, this method is not only incredibly time-consuming for large DNNs, but also infeasible in cases where access to the original training set is restricted. Additionally, retraining the DNN does not guarantee correctness on the faulty inputs added to the new training set. Another pointwise repair approach is to *fine tune* the DNN on the identified faulty inputs using gradient descent. This strategy, however, significantly increases the risk of high drawdown [Kemker et al. 2018]. A less desirable option is to simply accept the faulty behavior of the DNN as a natural consequence of deep learning. This can be incredibly dangerous in safety-critical applications of DNNs, and the faulty behavior can even be exploited by bad actors if left unpatched.

There are many scenarios in which we might want to specify how the DNN should behave for an *infinite* set of inputs. One approach to represent such an infinite set of points is via *V-polytopes*, where a vertex representation of a convex bounded polytope is used. *V-polytope repair of DNNs* takes as input a finite set of V-polytopes and the specification of the corresponding behavior constraining the (infinite) set of points in each of the V-polytopes. For example, a V-polytope repair specification can be used to express the constraint that certain input regions of an aircraft collision avoidance DNN should satisfy some safety properties. Retraining and fine tuning only take finite sets of input points, so they cannot be classified as V-polytope repair methods. On the other hand, V-polytope repair can be reduced to pointwise repair if the V-polytope input is defined as a single vertex.

Recently, there have been a number of approaches addressing *Provable Repair* of DNNs [Fu and Li 2022; Goldberger et al. 2020; Sotoudeh and Thakur 2019, 2021b]. Provable Repair ensures that the resulting repaired DNN is guaranteed to satisfy the given repair specification (property **P3**).

In this paper, we present a new Provable Repair approach that, to the best of our knowledge, is the first method that includes *all* of the following features:

F1 *Architecture-Preserving*: Our method preserves the architecture of the DNN throughout the repair process. The repaired DNN will have an identical structure to the DNN pre-repair.
F2 *Arbitrary V-Polytopes*: Our method supports the repair of arbitrary V-polytopes.
F3 *Multi-Layer Repair*: Our approach supports the modification of the weights of multiple layers of a DNN rather than restricting edits to a single layer.
F4 *Non-Piecewise-Linear Activation Function Support*: Our approach is applicable to DNNs that use activation functions that have some linear pieces such as ReLU and Hardswish.
F5 *Efficiency*: Our repair algorithm runs in polynomial time.
F6 *Scalability*: Our method repairs large real-world DNNs, including ImageNet DNNs like VGG19 and ResNet152, and handles complex global properties, such as for the ACAS Xu networks.

Organization. § 2 presents key definitions and notation. § 3 illustrates the key insights of our architecture-preserving provable V-polytope repair method on a simple DNN. § 4 describes our algorithm in detail. § 5 presents a qualitative comparison between our approach and prior provable-repair techniques such as PRDNN [Sotoudeh and Thakur 2021b] and REASSURE [Fu and Li 2022]. § 6 outlines the implementation of our approach in a tool called APRNN. We evaluate APRNN using MNIST, ImageNet, and ACAS Xu networks and compare against PRDNN, REASSURE [Fu and Li 2023], as well as lookup-based approaches. § 7 shows that APRNN outperforms these prior techniques in terms of efficiency, scalability, and generalization, all while preserving the original network architecture. § 8 describes related work and § 9 concludes.

2 PRELIMINARIES

This section lists concepts and notations used in the rest of the paper.

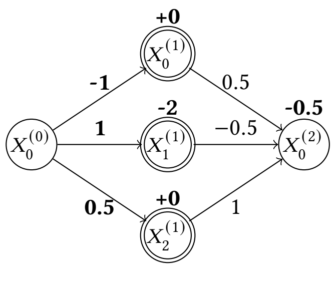
2.1 Deep Neural Networks

We restrict ourselves to fully-connected DNNs when describing our approach. However, our approach works for other DNN architectures as demonstrated by our experimental evaluation (§ 7). We use x, y, w to denote scalar values, and X, Y, W, and θ to denote matrix values.

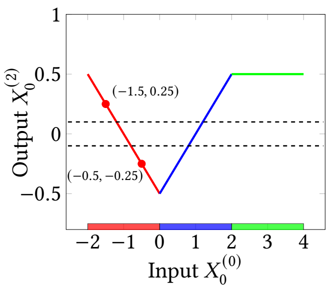
Definition 2.1. A *deep neural network (DNN)* $\mathcal{N}$ with L layers and parameters θ is a sequence of tuples $(W^{(0)}, B^{(0)}, \sigma^{(0)}), \cdots, (W^{(L-1)}, B^{(L-1)}, \sigma^{(L-1)})$ with $W^{(\ell)}, B^{(\ell)} \in \theta$. For the ℓth layer, $n^{(\ell)}$ and $n^{(\ell+1)}$ are the number of input and output neurons, respectively, $W^{(\ell)} \in \mathbb{R}^{n^{(\ell)} \times n^{(\ell+1)}}$ is the *weight* matrix, and $B^{(\ell)} \in \mathbb{R}^{n^{(\ell+1)}}$ is the *bias* matrix, and $\sigma^{(\ell)} : \mathbb{R}^{n^{(\ell+1)}} \to \mathbb{R}^{n^{(\ell+1)}}$ is an *activation function.*

Definition 2.2. The *identity activation function is a vector-valued function* $\mathtt{Id} : \mathbb{R}^n \to \mathbb{R}^n$ *defined component-wise as* $\mathtt{Id}(X)_i = X_i$.

Definition 2.3. The *ReLU activation function is a vector-valued function* $\mathtt{ReLU} : \mathbb{R}^n \to \mathbb{R}^n$ *defined component-wise as* $\mathtt{ReLU}(X)_i = \mathrm{relu}(X_i)$ where $\mathrm{relu}(x) = x$ if $x \geq 0$, otherwise $\mathrm{relu}(x) = 0$.



(a) DNN $\mathcal{N}_1$.



(b) Input-output plot of $\mathcal{N}_1$.

Fig. 5. DNN $\mathcal{N}_1$ and its input-output plot. The nodes $X_i^{(1)}$ have ReLU activation functions.

Definition 2.4. The *Hardswish activation function* is a vector-valued function $\texttt{Hardswish} : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ defined component-wise as $\texttt{Hardswish}(X)_i = \text{hardswish}(X_i)$ where $\text{hardswish}(x) = 0$ if $x \leq -3$, $\text{hardswish}(x) = x$ if $x \geq 3$, otherwise $\text{hardswish}(x) = x \cdot (x+3)/6$.

Example 2.5. Figure 5(a) shows a graphical representation of the DNN $\mathcal{N}_1$; weights are shown on arrows, biases are shown above nodes, and double nodes denote the ReLU activation function. The DNN $\mathcal{N}_1$ has two layers; i.e., $\texttt{len}(\mathcal{N}_1) = 2$. The first layer is $(W^{(0)}, B^{(0)}, \sigma^{(0)})$, where $W^{(0)} = \begin{bmatrix}-1 & 1 & 0.5\end{bmatrix}$, $B^{(0)} = \begin{bmatrix}0 & -2 & 0\end{bmatrix}$, $\sigma^{(0)} = \texttt{ReLU}$. The second layer is $(W^{(1)}, B^{(1)}, \sigma^{(1)})$, where $W^{(1)} = \begin{bmatrix}0.5 & -0.5 & 1\end{bmatrix}^\mathsf{T}$, $B^{(1)} = \begin{bmatrix}-0.5\end{bmatrix}$, $\sigma^{(1)} = \texttt{Id}$.

Definition 2.6. Given a DNN $\mathcal{N}$ with L layers and parameters θ and an input $X \in \mathbb{R}^{n^{(0)}}$, the network output $X^{(L)} \overset{\text{def}}{=} \mathcal{N}^\theta(X^{(0)})$, where $X^{(0)} \overset{\text{def}}{=} X$, and $X^{(\ell+1)} \overset{\text{def}}{=} \sigma^{(\ell)}(X^{(\ell)}W^{(\ell)} + B^{(\ell)})$. We use $\mathcal{N}(X)$ if the parameters θ are clear from the context.

Example 2.7. Consider the DNN $\mathcal{N}_1$ in Figure 5(a) discussed in Example 2.5. Given an input point $X^{(0)} = \begin{bmatrix}4\end{bmatrix}$, the output $X^{(1)}$ of layer 0 is $X^{(1)} = \begin{bmatrix}4\end{bmatrix}\begin{bmatrix}-1 & 1 & 0.5\end{bmatrix} + \begin{bmatrix}0 & -2 & 0\end{bmatrix} = \begin{bmatrix}-4 & 2 & 2\end{bmatrix}$ and the output $X^{(2)}$ of layer 1 is $X^{(2)} = \begin{bmatrix}0 & 2 & 2\end{bmatrix}\begin{bmatrix}0.5 & -0.5 & 1\end{bmatrix}^\mathsf{T} + \begin{bmatrix}-0.5\end{bmatrix} = \begin{bmatrix}0.5\end{bmatrix}$. Figure 5(b) plots of the output of $\mathcal{N}_1$ for the input range $[-2, 4]$.

We use $\text{accuracy}(\mathcal{N}, S)$ to denote the accuracy of the DNN $\mathcal{N}$ on a set S.

Definition 2.8. Given a DNN $\mathcal{N}$ with L layers, $\mathcal{N}^{(\ell_0:\ell_1)}$ denotes the slice of $\mathcal{N}$ from ℓ_0th to the ℓ_1–1th layer, where $\mathcal{N}^{(\ell_0:\ell_1)} \overset{\text{def}}{=} \left(W^{(\ell_0)}, B^{(\ell_0)}, \sigma^{(\ell_0)}\right), \left(W^{(\ell_0+1)}, B^{(\ell_0+1)}, \sigma^{(\ell_0+1)}\right), \ldots, \left(W^{(\ell_1-1)}, B^{(\ell_1-1)}, \sigma^{(\ell_1-1)}\right)$.

The *vertex representation (or V-representation)* of a bounded convex polytope defines the polytope as the convex hull of a finite set of vertices of the polytope. We use $\texttt{ConvexHull}(P)$ to denote the convex hull of the finite set of points P, and *V-polytope* to denote this finite set of points defining a bounded convex polytope.

Definition 2.9 (V-Polytope). A *V-polytope* P is defined as a finite set of points $\{{}_0X, {}_1X, \ldots, {}_nX\}$, ${}_iX \in \mathbb{R}^n$. The V-polytope P represents the bounded convex polytope $\texttt{ConvexHull}(P)$.

An alternative representation of convex polytopes is by using an *H-representation*, which represents a polytope as a set of linear constraints. Converting a polytope from V-representation to H-representation is called the facet enumeration problem, and from H-representation to V-representation is called the vertex enumeration problem. It is not known whether there exists a polynomial time algorithm in general for either of these problems [Bremner et al. 1997]. In the rest of the paper, we use "polytope" to mean V-polytope, and use "affine" and "linear" interchangeably.

$\widehat{x}, \widehat{y}, \widehat{w}$ denotes symbolic variables, $\widehat{X}, \widehat{Y}, \widehat{W}, \widehat{\theta}$ denotes vectors of symbolic variables. $\mathcal{E}_{\widehat{V}}$ and $\mathcal{F}_{\widehat{V}}$ denote the set of symbolic expressions and formulas over variables $\widehat{V}$, respectively.

Definition 2.10. A *linear expression* is of the form $\Sigma_i w_i \widehat{x}_i + \widehat{y}$. A *linear formula* is of the form $\bigwedge_i \Sigma_j w_j \widehat{x}_j + \widehat{y}_i \bowtie 0$, where $\bowtie \in \{<, \leq, =, \geq, >\}$.

$\psi(\widehat{x}_1, \widehat{x}_2, \ldots, \widehat{x}_n)$ denotes a formula over variables $\widehat{x}_i$. We overload this notation and use $\psi(\widehat{E}_1, \widehat{E}_2, \ldots, \widehat{E}_n)$ to denote a formula $\psi(\widehat{x}_1, \widehat{x}_2, \ldots, \widehat{x}_n) \wedge \bigwedge_i \widehat{x}_i = \widehat{E}_i$. Given $\widehat{E} \in \mathcal{E}_{\widehat{V}}$, $[\![\widehat{E}]\!]_V$ denotes the valuation of the expression $\widehat{E}$ using the value V. We define a *symbolic V-polytope* $\widehat{P} = \left\{{}_0\widehat{X}, \cdots, {}_n\widehat{X}\right\}$ as a set of symbolic points ${}_i\widehat{X} \in \mathcal{E}^m_{\widehat{V}}$. We use $[\![\widehat{\mathcal{P}}]\!]_V$ to denote $\left\{[\![{}_0\widehat{X}]\!]_V, \cdots, [\![{}_n\widehat{X}]\!]_V\right\}$.

2.2 Provable Repair

Given a DNN $\mathcal{N}$ and a repair specification, the goal of provable repair is to find a DNN $\mathcal{N}'$ that satisfies the given repair specification. Provable Repair approaches can be classified along two axes: (a) the type of repair specifications (pointwise vs. polytope), and (b) whether or not the approach modifies the original architecture of the DNN $\mathcal{N}$ (architecture-modifying vs. architecture preserving). Before defining these four variants of the problem, we define two quantitative metrics used to evaluate a repair technique: *drawdown* and *generalization.*

The *Drawdown set* is a set of points that are disjoint from the repair specification and are representative of the existing knowledge in the DNN. The *Generalization set* is a set of points that are disjoint from but similar to those in the repair specification.

Definition 2.11. Given a drawdown set S and DNNs $\mathcal{N}$ and $\mathcal{N}'$, the *drawdown* of $\mathcal{N}'$ with respect to $\mathcal{N}$ is defined as accuracy$(\mathcal{N}, S)$ − accuracy$(\mathcal{N}', S)$. Lower drawdown is better.

Definition 2.12. Given a generalization set S and DNNs $\mathcal{N}$ and $\mathcal{N}'$, the *generalization* of $\mathcal{N}'$ with respect to $\mathcal{N}$ is defined as accuracy$(\mathcal{N}', S)$ − accuracy$(\mathcal{N}, S)$. Higher generalization is better.

Definition 2.13. A *pointwise repair specification* is a tuple $(\mathcal{X}, \Psi)$, where $\mathcal{X} \overset{\text{def}}{=} \{{}_1X, {}_2X, \ldots, {}_nX\}$ and $\Psi \overset{\text{def}}{=} \{\psi_1, \psi_2, \ldots, \psi_n\}$. A *DNN* $\mathcal{N}^\theta$ *satisfies* $(\mathcal{X}, \Psi)$ iff the following formula is true: $\bigwedge_{1 \leq i \leq n} \psi_i\big(\mathcal{N}^\theta({}_iX)\big)$.

Definition 2.14. Given a DNN $\mathcal{N}$ and a pointwise repair specification $(\mathcal{X}, \Psi)$, the *provable pointwise repair problem* is to find a DNN $\mathcal{N}'$ that satisfies $(\mathcal{X}, \Psi)$. The architecture of $\mathcal{N}'$ need not be the same as that of $\mathcal{N}$.

Definition 2.15. Given a DNN $\mathcal{N}^\theta$ and a pointwise repair specification $(\mathcal{X}, \Psi)$, the *architecture-preserving provable pointwise repair problem* is to find parameters θ' such that $\mathcal{N}^{\theta'}$ satisfies $(\mathcal{X}, \Psi)$.

Definition 2.16. A *V-polytope repair specification* is a tuple $(\mathcal{P}, \Psi)$, where $\mathcal{P} \overset{\text{def}}{=} \{P_1, P_2, \ldots, P_n\}$ with P_i being a V-polytope, and $\Psi \overset{\text{def}}{=} \{\psi_1, \psi_2, \ldots, \psi_n\}$ where $\psi_i(X)$ is a linear formula. A *DNN* $\mathcal{N}^\theta$ *satisfies* $(\mathcal{P}, \Psi)$ iff the following formula is true: $\bigwedge_{1 \leq i \leq n} \forall X \in \texttt{ConvexHull}(P_i). \psi_i\big(\mathcal{N}^\theta(X)\big)$.

Definition 2.17. Given a DNN $\mathcal{N}$ and a V-polytope repair specification $(\mathcal{P}, \Psi)$, the *provable V-polytope repair problem* is to find a DNN $\mathcal{N}'$ that satisfies $(\mathcal{P}, \Psi)$. The architecture of $\mathcal{N}'$ need not be the same as that of $\mathcal{N}$.

Definition 2.18. Given a DNN $\mathcal{N}^\theta$ and a V-polytope repair specification $(\mathcal{P}, \Psi)$, the *architecture-preserving provable V-polytope repair problem* is to find parameters θ' such that $\mathcal{N}^{\theta'}$ satisfies $(\mathcal{P}, \Psi)$.

3 OVERVIEW

This section illustrates our architecture-preserving provable V-polytope repair algorithm using a simple DNN $\mathcal{N}_1$ (Figure 5(a)). We highlight the key insights of our approach using a pointwise repair specification (§3.1), before moving onto a V-polytope repair specification (§3.2). We elide details in favor of exposition, deferring them to §4. The examples in this section only use fully-connected linear layers and the ReLU activation function, which is piecewise-linear. However, our formalism described in §4 handles any activation functions that have some linear pieces, such as Hardswish (Definition 2.4). Furthermore, our implementation (§6) also handles convolutional layers, pooling layers, and residual layers, as demonstrated by our experimental evaluation (§7).

3.1 Provable Pointwise Repair

Consider the pointwise repair specification $(\mathcal{X}, \Psi)$, where $\mathcal{X} \overset{\text{def}}{=} \{ [-1.5], [-0.5] \}$, $\Psi \overset{\text{def}}{=} \{\psi_1, \psi_1\}$ and $\psi_1(\widehat{\boldsymbol{Y}}) \overset{\text{def}}{=} -0.1 \le \widehat{\boldsymbol{Y}}_0 \le 0.1$. The DNN $\mathcal{N}_1$ does not satisfy $(\mathcal{X}, \Psi)$: $\psi_1\left(\mathcal{N}_1([-1.5])\right)$ and $\psi_1\left(\mathcal{N}_1([-0.5])\right)$ are both false, because $\mathcal{N}_1([-1.5])_0 = 0.25 > 0.1$, $\mathcal{N}_1([-0.5])_0 = -0.25 < -0.1$.

Let $\widehat{\boldsymbol{\theta}}$ represent the symbolic variables corresponding to each of the parameters θ in the DNN $\mathcal{N} : \mathbb{R}^m \to \mathbb{R}^n$. Let $\mathcal{N}^{\widehat{\boldsymbol{\theta}}}(X) \in \mathcal{E}^n_{\widehat{\boldsymbol{\theta}}}$ denote the symbolic expression representing the output of $\mathcal{N}$ for input $X \in \mathbb{R}^m$. Using this notation, we can phrase our architecture-preserving provable pointwise repair problem for $\mathcal{N}_1$ as finding a satisfying assignment θ' for the following formula $\phi_{\text{spec}} \in \mathcal{F}_{\widehat{\boldsymbol{\theta}}}$:

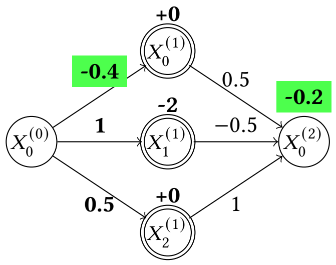
$$\phi_{\text{spec}} \overset{\text{def}}{=} -0.1 \le \mathcal{N}_1^{\widehat{\boldsymbol{\theta}}}([-1.5])_0 \le 0.1 \land -0.1 \le \mathcal{N}_1^{\widehat{\boldsymbol{\theta}}}([-0.5])_0 \le 0.1 \tag{1}$$

In general, the expression $\mathcal{N}^{\widehat{\boldsymbol{\theta}}}(X)$ will be highly complex and non-linear, involving quadratic terms (if the DNN has more than a single layer), disjunctions (to model piecewise-linear functions such as ReLU), or other non-linear functions (due to activation functions such as Hardswish). Thus, directly solving Equation 1 is impractical in most cases.

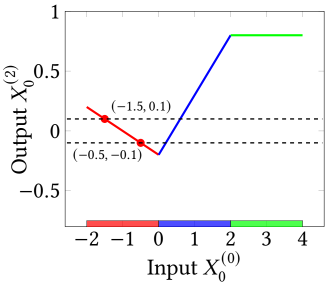
One observation made by prior approaches [Goldberger et al. 2020; Sotoudeh and Thakur 2021b] is that if the repair is restricted to only modify parameters in a single layer, then the quadratic terms can be avoided in $\mathcal{N}^{\widehat{\boldsymbol{\theta}}}(X)$. In our approach, we **extend this observation** as follows: the quadratic terms can be avoided if we restrict the repair to only modify weights in some single layer k but allow modification of *biases in all layers* $\ell \ge k$. Though simple in hindsight, allowing more biases to be modified by the repair improves the quality of the repair in practice. In our example, we restrict the repair to only modify first layer weights and all layers' biases; that is, $\widehat{\boldsymbol{W}}^{(0)}$, $\widehat{\boldsymbol{B}}^{(0)}$, and $\widehat{\boldsymbol{B}}^{(1)}$ are the symbolic variables corresponding to their respective weights and biases in $\mathcal{N}_1$. However, even this restriction is not sufficient to make the repair problem tractable. The problem remains NP-hard even when the repair is restricted to only modify the parameters of a single layer and the DNN contains only ReLU activation functions [Goldberger et al. 2020].

The **first key insight** of our approach is to find a *linear formula* ϕ_{lin} *that implies* ϕ_{spec}. Any satisfying assignment θ' of ϕ_{lin} will imply that $\mathcal{N}_1^{\theta'}$ satisfies the repair specification. Checking satisfiability of the linear formula can be done in polynomial time [Khachiyan 1979] making our repair approach efficient and scalable to large DNNs. Furthermore, a linear programming (LP) solver can be used to find a θ' that minimizes, for instance, the difference between θ' and the original parameters θ [Gurobi Optimization, LLC 2022], which lowers repair drawdown (Definition 2.11).

The **key primitive** we define is a *conditional symbolic function* $\widetilde{\sigma}$ corresponding to an activation function σ (Definition 4.3). We illustrate this concept for the ReLU activation function using the DNN $\mathcal{N}_1$. Consider the input point $X^{(0)} = [-1.5]$, then $\widehat{\boldsymbol{X}}^{(1)}_0$, the first symbolic output of layer 0, is $\texttt{ite}(-1.5\widehat{\boldsymbol{W}}^{(0)}_{0,0} + \widehat{\boldsymbol{B}}^{(0)}_0 \ge 0, -1.5\widehat{\boldsymbol{W}}^{(0)}_{0,0} + \widehat{\boldsymbol{B}}^{(0)}_0, 0)$. Notice that in DNN $\mathcal{N}_1$, we have $-1.5W^{(0)}_{0,0} + B^{(0)}_0 \ge 0$.



(a) DNN $\mathcal{N}_2$.



(b) Input-output plot of $\mathcal{N}_2$.

Fig. 6. DNN $\mathcal{N}_2$ that satisfies the pointwise repair specification $(\mathcal{X}, \Psi)$ in §3.1.

If we constrain the values of $\widehat{W}^{(0)}_{0,0}$ and $\widehat{B}^{(0)}_0$ such that $\varphi_1 \overset{\text{def}}{=} -1.5\widehat{W}^{(0)}_{0,0} + \widehat{B}^{(0)}_0 \geq 0$ is true, then $\widehat{X}^{(1)}_0 = -1.5\widehat{W}^{(0)}_{0,0} + \widehat{B}^{(0)}_0$. Similarly, if we constrain the values of $\widehat{W}^{(0)}_{0,1}$ and $\widehat{B}^{(0)}_1$ such that $\varphi_2 \overset{\text{def}}{=} -1.5\widehat{W}^{(0)}_{0,1} + \widehat{B}^{(0)}_1 < 0$ is true, then $\widehat{X}^{(1)}_1 = 0$. If we constrain the values of $\widehat{W}^{(0)}_{0,2}$ and $\widehat{B}^{(0)}_2$ such that $\varphi_3 \overset{\text{def}}{=} -1.5\widehat{W}^{(0)}_{0,2} + \widehat{B}^{(0)}_2 < 0$ is true, then $\widehat{X}^{(1)}_2 = 0$. Using this symbolic value for $\widehat{X}^{(1)}$, we get $\widehat{X}^{(2)} = \left[-0.75\widehat{W}^{(0)}_{0,0} + 0.5\widehat{B}^{(0)}_0 + \widehat{B}^{(1)}_0\right]$. We can confirm that the linear formula

$$\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge -0.1 \leq -0.75\widehat{W}^{(0)}_{0,0} + 0.5\widehat{B}^{(0)}_0 + \widehat{B}^{(1)}_0 \leq 0.1$$

implies $-0.1 \leq \mathcal{N}^{\widehat{\theta}}_1\left(\left[-1.5\right]\right)_0 \leq 0.1$. Similarly, for input $X^{(0)} = \left[-0.5\right]$, we get the linear formula

$$-0.5\widehat{W}^{(0)}_{0,0} + \widehat{B}^{(0)}_0 \geq 0 \wedge -0.5\widehat{W}^{(0)}_{0,1} + \widehat{B}^{(0)}_1 < 0 \wedge -0.5\widehat{W}^{(0)}_{0,2} + \widehat{B}^{(0)}_2 < 0 \wedge -0.1 \leq -0.25\widehat{W}^{(0)}_{0,0} + 0.5\widehat{B}^{(0)}_0 + \widehat{B}^{(1)}_0 \leq 0.1$$

which implies $-0.1 \leq \mathcal{N}^{\widehat{\theta}}_1\left(\left[-0.5\right]\right)_0 \leq 0.1$.

Those constraints corresponding to each of the points in the repair specification formulates an LP problem. To find the minimal modification of $\mathcal{N}_1$, we minimize the L^∞ and normalized L^1 norm of the delta of both the parameters and outputs. The DNN $\mathcal{N}_2$ (Figure 6(a)) shows this minimal repair of $\mathcal{N}_1$ that satisfies the given pointwise repair specification $(\mathcal{X}, \Psi)$.

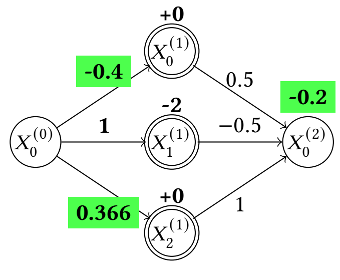
3.2 Provable V-Polytope Repair

Consider the V-polytope repair specification $(\mathcal{P}_1, \Psi_1)$ where $\mathcal{P}_1 \overset{\text{def}}{=} \{P_1\}$ with the V-polytope $P_1 \overset{\text{def}}{=} \left\{\left[-1.5\right], \left[-0.5\right]\right\}$, and $\Psi_1 \overset{\text{def}}{=} \{\psi_1\}$ where $\psi_1(\widehat{Y}) \overset{\text{def}}{=} -0.1 \leq \widehat{Y}_0 \leq 0.1$. The DNN $\mathcal{N}_1$ does not satisfy the V-polytope specification $(\mathcal{P}_1, \Psi_1)$; viz., the following formula is not true:

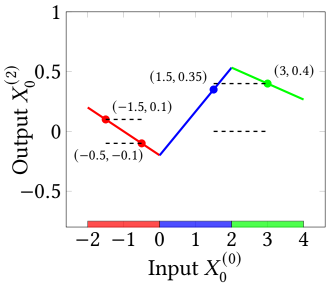
$$\forall X \in \texttt{ConvexHull}\left(\left\{\left[-1.5\right], \left[-0.5\right]\right\}\right). -0.1 \leq \mathcal{N}_1(X)_0 \leq 0.1 \tag{2}$$

This V-polytope repair specification is an extension of the pointwise repair specification $(\mathcal{X}, \Psi)$ we discussed in § 3.1: the pointwise repair specification constrained the behavior of only the two points $\left[-1.5\right]$ and $\left[-0.5\right]$, while the V-polytope repair specification constrains the behavior of $\mathcal{N}_1$ on all (infinite) points in $\texttt{ConvexHull}\left(\left\{\left[-1.5\right], \left[-0.5\right]\right\}\right)$. However, notice that the DNN $\mathcal{N}_2$ (Figure 6(a)) repaired using the pointwise repair specification $(\mathcal{X}, \Psi)$ also satisfies the V-polytope repair specification $(\mathcal{P}_1, \Psi_1)$.

Based on this observation, one might be tempted to conclude that merely repairing the vertices of a V-polytope is sufficient to repair all points in the convex hull of the vertices. The next example shows this not to be true. Consider the V-polytope repair specification $(\mathcal{P}_2, \Psi_2)$ where $\mathcal{P}_2 \overset{\text{def}}{=} \{P_1, P_2\}$ with the V-polytope $P_1 \overset{\text{def}}{=} \left\{\left[-1.5\right], \left[-0.5\right]\right\}$, $P_2 \overset{\text{def}}{=} \left\{\left[1.5\right], \left[3\right]\right\}$, and $\Psi_2 \overset{\text{def}}{=} \{\psi_1, \psi_2\}$ where $\psi_1(\widehat{Y}) \overset{\text{def}}{=}$



(a) DNN $\mathcal{N}_3$.



(b) Input-output plot of DNN $\mathcal{N}_3$.

Fig. 7. DNN $\mathcal{N}_3$ that does not satisfy the V-polytope repair specification $(\mathcal{P}_2, \Psi_2)$ in §3.2.

$-0.1 \le \widehat{Y}_0 \le 0.1$, $\psi_2(\widehat{Y}) \overset{\text{def}}{=} 0 \le \widehat{Y}_0 \le 0.4$. The DNN $\mathcal{N}_1$ does not satisfy this repair specification, because the following formula is not true:

$$\begin{aligned} &\forall X^{(0)} \in \texttt{ConvexHull}(\{[-1.5],[-0.5]\}).\ -0.1 \le \mathcal{N}_1(X^{(0)})_0 \le 0.1\ \wedge \\ &\forall X^{(0)} \in \texttt{ConvexHull}(\{\ [1.5]\ ,\ \ [3]\ \ \}).\quad 0 \quad \le \mathcal{N}_1(X^{(0)})_0 \le 0.4 \end{aligned}$$

Suppose we just repair the vertices of the polytopes so that the following formula is true:

$$\begin{aligned} -0.1 \le \mathcal{N}_1([-1.5])_0 \le 0.1\ \wedge\ -0.1 \le \mathcal{N}_1([-0.5])_0 \le 0.1\ \wedge \\ 0 \le \mathcal{N}_1(\ [1.5]\ )_0 \le 0.4\ \wedge \qquad 0 \le \mathcal{N}_1(\ \ [3]\ \ )_0 \le 0.4 \end{aligned}$$

The DNN $\mathcal{N}_3$ in Figure 7(a) shows the corresponding repaired DNN. As we can see in Figure 7(b), DNN $\mathcal{N}_3$ does not satisfy the V-polytope specification $(\mathcal{P}_2, \Psi_2)$. In particular, though the DNN $\mathcal{N}_3$ satisfies the specification ψ_1 corresponding to the V-polytope $P_1 \overset{\text{def}}{=} \{[-1.5], [-0.5]\}$, it does not satisfy the specification ψ_2 corresponding to the V-polytope $P_2 \overset{\text{def}}{=} \{[1.5], [3]\}$. For instance, $\mathcal{N}_3([2])_0 = 0.532 > 0.4$, which violates the specification ψ_2.

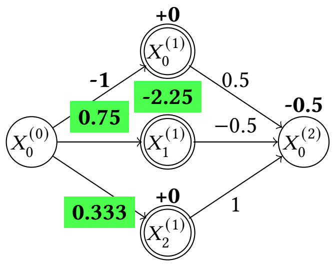
What property distinguishes P_1 and P_2 for the DNNs $\mathcal{N}_1$ and $\mathcal{N}_3$? $\mathcal{N}_1$ and $\mathcal{N}_3$ are *locally linear* (Definition 4.1) for P_1; that is, there exists linear functions f_1 and f_2 such that $\mathcal{N}_1(X) = f_1(X)$ and $\mathcal{N}_3(X) = f_2(X)$ for all $X \in \texttt{ConvexHull}(P_1)$. This leads us to the **second key insight** of our approach: if a DNN $\mathcal{N}$ is locally linear for a V-polytope P, then repairing the behavior of $\mathcal{N}$ on the vertices of P using our approach is sufficient to guarantee that $\mathcal{N}$ satisfies a V-polytope specification over P. This insight explains the behavior for P_1 in the above example. However, this insight does not directly indicate how to tackle the V-polytope P_2 for which $\mathcal{N}_1$ and $\mathcal{N}_3$ are not locally linear.

The **third key insight** of our approach is that the parameters of a DNN $\mathcal{N}$ can be modified so that $\mathcal{N}$ is locally linear for a given V-polytope P. Consider again the DNN $\mathcal{N}_1$ and the V-polytope repair specification $(\mathcal{P}_2, \Psi_2)$. Consider the following constraints:

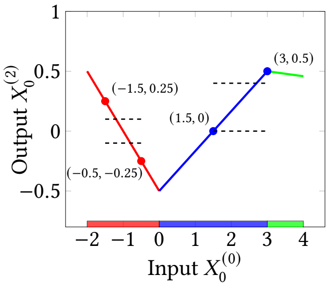
$$\begin{array}{llll} -1.5\widehat{W}^{(0)}_{0,0} + \widehat{B}^{(0)}_0 \ge 0\ \wedge & -1.5\widehat{W}^{(0)}_{0,1} + \widehat{B}^{(0)}_1 < 0\ \wedge & -1.5\widehat{W}^{(0)}_{0,2} + \widehat{B}^{(0)}_2 < 0\ \wedge \\ -0.5\widehat{W}^{(0)}_{0,0} + \widehat{B}^{(0)}_0 \ge 0\ \wedge & -0.5\widehat{W}^{(0)}_{0,1} + \widehat{B}^{(0)}_1 < 0\ \wedge & -0.5\widehat{W}^{(0)}_{0,2} + \widehat{B}^{(0)}_2 < 0\ \wedge \\ 1.5\widehat{W}^{(0)}_{0,0} + \widehat{B}^{(0)}_0 < 0\ \wedge & 1.5\widehat{W}^{(0)}_{0,1} + \widehat{B}^{(0)}_1 < 0\ \wedge & 1.5\widehat{W}^{(0)}_{0,2} + \widehat{B}^{(0)}_2 \ge 0\ \wedge \\ 3\widehat{W}^{(0)}_{0,0} + \widehat{B}^{(0)}_0 < 0\ \wedge & 3\widehat{W}^{(0)}_{0,1} + \widehat{B}^{(0)}_1 < 0\ \wedge & 3\widehat{W}^{(0)}_{0,2} + \widehat{B}^{(0)}_2 \ge 0 \end{array} \tag{3}$$

By solving the above linear formula we get the corresponding DNN $\mathcal{N}_4$ shown in Figure 8(a). As shown in its input-output plot Figure 8(b), the DNN $\mathcal{N}_4$ is locally linear for both P_1 and P_2.

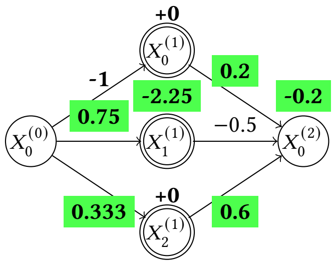
Now we can modify the parameters of $\mathcal{N}_4$ to repair the behavior on just the vertices of P_1 and P_2. We could again choose to modify the first-layer weights and all biases. However, for illustrative purposes, we will choose to modify the weights and biases of only the second layer of $\mathcal{N}_4$. This



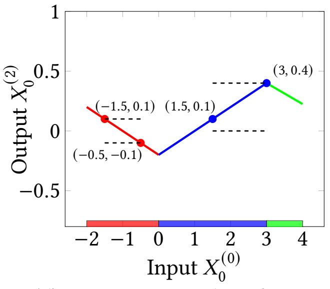
(a) DNN $\mathcal{N}_4$



(b) Input-output plot of $\mathcal{N}_4$.



(c) DNN $\mathcal{N}_5$.



(d) Input-output plot of $\mathcal{N}_5$.

Fig. 8. DNN $\mathcal{N}_4$ that is locally linear for V-polytopes P_1 and P_2 from $\mathcal{P}_2$ in §3.2, as well as DNN $\mathcal{N}_5$ that satisfies the V-polytope repair specification $(\mathcal{P}_2, \Psi_2)$ in §3.2.

leads us to the following linear formula:

$$\begin{aligned} -0.1 \leq 1.5\widehat{W}^{(1)}_{0,0} + \widehat{B}^{(1)}_{0} &\leq 0.1 \ \wedge \ -0.1 \leq 0.5\widehat{W}^{(1)}_{0,0} + \widehat{B}^{(1)}_{0} \leq 0.1 \ \wedge \\ 0 \leq 0.5\widehat{W}^{(1)}_{2,0} + \widehat{B}^{(1)}_{0} &\leq 0.4 \ \wedge \quad\ \ 0 \leq \quad \widehat{W}^{(1)}_{2,0} + \widehat{B}^{(1)}_{0} \leq 0.4 \end{aligned}$$

Figure 8(c) presents the resulting repaired DNN $\mathcal{N}_5$. As can be seen in its input-output plot Figure 8(d), $\mathcal{N}_5$ satisfies the V-polytope repair specification $(\mathcal{P}_2, \Psi_2)$. Comparing $\mathcal{N}_1$ and $\mathcal{N}_5$, we see that our repair approach modified *weights across multiple layers and biases across multiple layers.*

4 APPROACH

This section describes our architecture-preserving provable V-polytope repair algorithm (Algorithm 1). A pointwise repair specification (Definition 2.13) can be expressed as a V-polytope repair specification (Definition 2.16) in which each polytope is a single point. Consequently, an approach for provable V-polytope repair subsumes provable pointwise repair. Thus, we only discuss our algorithm for architecture-preserving provable V-polytope repair. Our experimental evaluation demonstrates that our tool handles both provable pointwise repair and provable V-polytope repair.

The function `VPolytopeRepair`$(\mathcal{N}, \mathcal{P}^{(0)}, \Psi, s, k)$ in Algorithm 1 takes as input a DNN $\mathcal{N}$, a V-polytope repair specification $(\mathcal{P}^{(0)}, \Psi)$ (Definition 2.16), a network partition s (Definition 4.13), and a layer index k. The function returns either $\bot$, indicating the algorithm was unable to repair the DNN $\mathcal{N}$ with the given arguments, or returns a repaired DNN $\mathcal{N}_{\texttt{ret}}$ that satisfies $(\mathcal{P}^{(0)}, \Psi)$. The algorithm modifies parameters in multiple layers as determined by s and k.

In the rest of the section, we describe the working of Algorithm 1 using examples, as well as a series of theorems and their proof sketches related to its correctness and efficiency. Detailed proofs can be found in the extended version [Tao et al. 2023]. We first introduce the concept of a function being *locally linear* for a polytope P, which is key to our approach.

Algorithm 1: $\texttt{VPolytopeRepair}(\mathcal{N}, \mathcal{P}^{(0)}, \Psi, s, k)$

```
Input: A DNN $\mathcal{N}$, a set of V-polytopes $\mathcal{P}$, a repair specification $\Psi$, a partition $s$, and a layer index $k$.
Output: Repaired DNN $\mathcal{N}_{\texttt{ret}}$ that satisfies $(\mathcal{P}, \Psi)$, or $\bot$.
$\mathcal{N}_{\texttt{ret}} \leftarrow \texttt{copy}(\mathcal{N})$
foreach $k_i, \ell_i \in s$ do
    $\mathcal{N}_{\texttt{ret}}^{(0:\ell_i)} \leftarrow \texttt{Shift\&Assert}(\mathcal{N}_{\texttt{ret}}^{(0:\ell_i)}, \mathcal{N}^{(0:\ell_i)}, \mathcal{P}^{(0)}, \top, k_i)$
    if $\mathcal{N}_{\texttt{ret}}^{(0:\ell_i)} = \bot$ then return $\bot$
return $\texttt{Shift\&Assert}(\mathcal{N}_{\texttt{ret}}, \mathcal{N}, \mathcal{P}^{(0)}, \Psi, k)$
def $\texttt{Shift\&Assert}(\mathcal{N}, \mathcal{N}_{\texttt{og}}, \mathcal{P}^{(0)}, \Psi, k)$ :
    $\mathcal{N}_{\texttt{ret}},\ \varphi,\ L \leftarrow \texttt{copy}(\mathcal{N}),\ \top,\ \texttt{len}(\mathcal{N})$
    foreach $P^{(k)} \in \mathcal{N}^{(0:k)}(\mathcal{P}^{(0)})$ do
        $\widehat{P}^{(L)}, \varphi_\sigma \leftarrow \widetilde{\mathcal{N}}^{(k:L)}(P^{(k)})$
        $\widehat{\mathcal{P}}^{(L)}, \varphi \leftarrow \texttt{append}(\widehat{\mathcal{P}}^{(L)}, \widehat{P}^{(L)}), \varphi \wedge \varphi_\sigma$
    $\varphi \leftarrow \varphi \wedge \Psi(\widehat{\mathcal{P}}^{(L)})$
    $\vec{\Delta} \leftarrow \left\langle \widehat{\mathcal{P}}^{(L)} - \mathcal{N}_{\texttt{og}}(\mathcal{P}^{(0)}) \;\middle|\; \widehat{\theta}_{\mathcal{N}^{(k:L)}} - \theta_{\mathcal{N}^{(k:L)}} \right\rangle$
    $\theta'_{\mathcal{N}^{(k:L)}} \leftarrow \texttt{Minimize}\ \|\vec{\Delta}\|_p\ \texttt{Subject To}\ \varphi$
    if $\theta'_{\mathcal{N}^{(k:L)}} = \bot$ then return $\bot$
    $\mathcal{N}_{\texttt{ret}}^{(k:L)} \leftarrow \texttt{Update}(\mathcal{N}_{\texttt{ret}}^{(k:L)}, \theta'_{\mathcal{N}^{(k:L)}})$
    return $\mathcal{N}_{\texttt{ret}}$
def $\widetilde{\mathcal{N}}(P^{(0)})$ :
    $\varphi,\ L,\ X_{\texttt{ref}}^{(0)} \leftarrow \top,\ \texttt{len}(\mathcal{N}),\ \texttt{calc\_ref}(P^{(0)})$
    foreach ${}_jX^{(0)} \in P^{(0)}$ do
        ${}_j\widehat{X}^{(L)}, {}_j\varphi_\sigma \leftarrow \widetilde{\mathcal{N}}[X_{\texttt{ref}}^{(0)}]({}_jX^{(0)})$
    return $\bigcup_j \{{}_j\widehat{X}^{(L)}\},\ \bigwedge_j {}_j\varphi_\sigma$
def $\widetilde{\mathcal{N}}[X_{\texttt{ref}}^{(0)}](X^{(0)})$ :
    $L \leftarrow \texttt{len}(\mathcal{N})$
    foreach $\ell \in \{0, \cdots, L-1\}$ do
        if $\ell = 0$ then $\widehat{X}_{\texttt{pre}}^{(\ell+1)} \leftarrow X^{(\ell)}\widehat{W}^{(\ell)} + \widehat{B}^{(\ell)}$
        else $\widehat{X}_{\texttt{pre}}^{(\ell+1)} \leftarrow \widehat{X}^{(\ell)}W^{(\ell)} + \widehat{B}^{(\ell)}$
        $X_{\texttt{ref}}^{(\ell+1)} \leftarrow X_{\texttt{ref}}^{(\ell)}W^{(\ell)} + B^{(\ell)}$
        $\widehat{X}^{(\ell+1)}, \varphi_\sigma^{(\ell+1)} \leftarrow \widetilde{\sigma}^{(\ell)}[X_{\texttt{ref}}^{(\ell+1)}](\widehat{X}_{\texttt{pre}}^{(\ell+1)})$
        $X_{\texttt{ref}}^{(\ell+1)} \leftarrow \sigma^{(\ell)}(X_{\texttt{ref}}^{(\ell+1)})$
    return $\widehat{X}^{(L)},\ \bigwedge_{1\le\ell\le L} \varphi_\sigma^{(\ell)}$
def $\widetilde{\texttt{ReLU}}[X_{\texttt{ref}}](\widehat{X})$ :
    $\widehat{Y},\ \varphi = \langle\rangle,\ \top$
    foreach $\widehat{X}_i, X_i^{\texttt{ref}} \in \texttt{zip}(\widehat{X}, X_{\texttt{ref}})$ do
        if $X_i^{\texttt{ref}} \ge 0$ then
            $\widehat{Y},\ \varphi \leftarrow \texttt{append}(\widehat{Y}, \widehat{X}_i),\ \varphi \wedge \widehat{X}_i \ge 0$
        else $\widehat{Y},\ \varphi \leftarrow \texttt{append}(\widehat{Y}, 0),\ \varphi \wedge \widehat{X}_i < 0$
    return $\widehat{Y},\ \varphi$
def $\widetilde{\texttt{Hardswish}}[X_{\texttt{ref}}](\widehat{X})$ :
    $\widehat{Y},\ \varphi = \langle\rangle,\ \top$
    foreach $\widehat{X}_i, X_i^{\texttt{ref}} \in \texttt{zip}(\widehat{X}, X_{\texttt{ref}})$ do
        if $X_i^{\texttt{ref}} \ge 0$ then
            $\widehat{Y},\ \varphi \leftarrow \texttt{append}(\widehat{Y}, \widehat{X}_i),\ \varphi \wedge \widehat{X}_i \ge 3$
        else $\widehat{Y},\ \varphi \leftarrow \texttt{append}(\widehat{Y}, 0),\ \varphi \wedge \widehat{X}_i \le -3$
    return $\widehat{Y},\ \varphi$
```

Definition 4.1. Given a function g and a V-polytope P, g is *locally linear* for the polytope P iff there exists a linear function f such that $g(X) = f(X)$ for all $X \in \texttt{ConvexHull}(P)$.

Intuitively, suppose a function g has linear pieces, then g being *locally linear* for the polytope P implies that P is entirely in a linear piece of g. DNNs that use piecewise-linear activation functions, such as ReLU, or activation functions that have linear pieces, such as Hardswish, have linear pieces.

Example 4.2. Consider the DNN $\mathcal{N}_1$ shown in Figure 5(a) and the corresponding input-output plot of Figure 5(b). For example, $\mathcal{N}_1$ is locally linear for V-polytopes $\{[-1.5], [-0.5]\}$, $\{[-2], [0]\}$, $\{[0], [2]\}$, and $\{[2], [4]\}$. $\mathcal{N}_1$ is locally linear for V-polytopes $\{[-1], [1]\}$ and $\{[1.5], [3]\}$.

Definition 4.3. For a given activation function $\sigma : \mathbb{R}^m \to \mathbb{R}^n$, a *conditional symbolic activation function* $\widetilde{\sigma}[X_{\texttt{ref}}] : \mathcal{E}^m_{\widehat{\theta}} \to \mathcal{E}^n_{\widehat{\theta}} \times \mathcal{F}_{\widehat{\theta}}$ with $X_{\texttt{ref}} \in \mathbb{R}^m$ satisfies the following conditions:

C1 If $\widehat{Y}, \varphi \overset{\text{def}}{=} \widetilde{\sigma}[X_{\texttt{ref}}](\widehat{X})$, then $\theta \vDash \varphi$ implies $[\![\widehat{Y}]\!]_\theta = \sigma([\![\widehat{X}]\!]_\theta)$.

C2 If $\widehat{Y}, \varphi \overset{\text{def}}{=} \widetilde{\sigma}[X_{\texttt{ref}}](\widehat{X})$ and $\widehat{X}$ is linear, then $\widehat{Y}$ is a linear expression and φ is a linear formula.

C3 Let the symbolic polytope $\widehat{P} \overset{\text{def}}{=} \left\{{}_0\widehat{X}, {}_1\widehat{X}, \dots, {}_{p-1}\widehat{X}, \right\}$, ${}_j\widehat{Y}, {}_j\varphi \overset{\text{def}}{=} \widetilde{\sigma}[X_{\texttt{ref}}]({}_j\widehat{X})$ for $0 \le j < p$, and $\varphi_{\widehat{P}} \overset{\text{def}}{=} \bigwedge_j {}_j\varphi$. If $\theta \vDash \varphi_{\widehat{P}}$, then σ is locally linear for the polytope $[\![\widehat{P}]\!]_\theta$.

Let $\widehat{Y}, \varphi \overset{\text{def}}{=} \widetilde{\sigma}[X_{\mathtt{ref}}](\widehat{X})$; the intuition behind a conditional symbolic activation function $\widetilde{\sigma}$ with a reference point $X_{\mathtt{ref}}$ is to *deterministically* constrain the symbolic input $\widehat{X}$ to a linear piece of σ using a linear formula φ such that the symbolic output $\widehat{Y}$ exactly encodes $\sigma(\widehat{X})$. Condition **C1** states that $[\![\widehat{Y}]\!]_\theta = \sigma([\![\widehat{X}]\!]_\theta)$ for any assignment θ that satisfies φ, which ensures the correctness of this conditional encoding of $\sigma(\widehat{X})$. Condition **C2** is necessary for formulating the repair problem as an LP problem; Condition **C1** and Condition **C2** imply that φ constrains $\widehat{X}$ to a linear piece of σ. Condition **C3** is necessary for provable polytope repair (Theorem 4.10-Condition **C3**). It implies that $\widetilde{\sigma}[X_{\mathtt{ref}}]$ constrains any input $\widehat{X}$ to the same linear piece as determined by $X_{\mathtt{ref}}$.

The following example demonstrates the conditional symbolic activation function $\widetilde{\mathtt{ReLU}}$ (Line 31 in Algorithm 1) for ReLU.

Example 4.4. Let $\widehat{Y}, \varphi \overset{\text{def}}{=} \widetilde{\mathtt{ReLU}}[X_{\mathtt{ref}}](\widehat{X})$ where $\widehat{X} \overset{\text{def}}{=} [\widehat{X}_0 \ \widehat{X}_1]$ and $X_{\mathtt{ref}} \overset{\text{def}}{=} [5 \ -2]$. We will show that $\widehat{Y} = [\widehat{X}_0 \ 0]$ and $\varphi = \widehat{X}_0 \geq 0 \wedge \widehat{X}_1 < 0$. On Line 33–36, $\widetilde{\mathtt{ReLU}}$ loops over each component i and constrains $\widehat{X}_i$ to the linear piece containing $X_i^{\mathtt{ref}}$. Recall that $\mathtt{ReLU}(X)_i = \text{relu}(X_i)$ is a piecewise linear function where $\text{relu}(x) = x$ if $x \geq 0$, otherwise $\text{relu}(x) = 0$.

S1a If $X_i^{\mathtt{ref}} \geq 0$, Line 35 constrains $\widehat{X}_i$ to the linear piece $\text{relu}(x) = x$ where $x \geq 0$ with $\widehat{X}_i \geq 0$, which implies $\widehat{Y}_i \overset{\text{def}}{=} \widehat{X}_i$. In this example, because $X_0^{\mathtt{ref}} = 5 \geq 0$, $\widehat{X}_0 \geq 0$ is in φ and $\widehat{Y}_0 \overset{\text{def}}{=} \widehat{X}_0$.

S1b If $X_i^{\mathtt{ref}} < 0$, Line 36 constrains $\widehat{X}_i$ to the linear piece $\text{relu}(x) = 0$ where $x < 0$ with $\widehat{X}_i < 0$, which implies $\widehat{Y}_i \overset{\text{def}}{=} 0$. In this example, because $X_1^{\mathtt{ref}} = -2 < 0$, $\widehat{X}_1 < 0$ is in φ and $\widehat{Y}_1 \overset{\text{def}}{=} 0$.

Theorem 4.5. $\widetilde{\mathtt{ReLU}}$ *is a conditional symbolic activation function for* ReLU.

Proof Sketch. Condition **C1** is satisfied because $\widehat{X}_i \geq 0$ implies $\widehat{Y}_i = \widehat{X}_i$ and $\widehat{X}_i < 0$ implies $\widehat{Y}_i = 0$ using the definition of ReLU. Condition **C2** is satisfied because $\widehat{X}_i \geq 0$, $\widehat{X}_i < 0$ are linear formulas and $\widehat{Y}_i = \widehat{X}_i$, $\widehat{Y}_i = 0$ are linear expressions if $\widehat{X}_i$ is a linear expression. Condition **C3** is satisfied because $\widetilde{\mathtt{ReLU}}[X_{\mathtt{ref}}]$ constrains any $\widehat{X}_i$ to the linear piece that contains $X_{\mathtt{ref}}$. Given $\theta \vDash \varphi_{\widehat{P}}$, $[\![\widehat{P}]\!]_\theta$ is entirely in the same linear piece, hence ReLU is locally linear for $[\![\widehat{P}]\!]_\theta$. □

Moreover, for a non-piecewise-linear activation function σ, there exists $\widetilde{\sigma}$ if σ has linear pieces. The following example demonstrates $\widetilde{\mathtt{Hardswish}}$ (Line 38 in Algorithm 1). The proof that $\widetilde{\mathtt{Hardswish}}$ is a conditional symbolic activation function can be found in the extended version [Tao et al. 2023].

Example 4.6. Let $\widehat{Y}, \varphi \overset{\text{def}}{=} \widetilde{\mathtt{Hardswish}}[X_{\mathtt{ref}}](\widehat{X})$ where $\widehat{X} \overset{\text{def}}{=} [\widehat{X}_0 \ \widehat{X}_1]$ and $X_{\mathtt{ref}} \overset{\text{def}}{=} [5 \ -2]$. We will show that $\widehat{Y} = [\widehat{X}_0 \ 0]$ and $\varphi = \widehat{X}_0 \geq 3 \wedge \widehat{X}_1 \leq -3$. On Line 40–43, $\widetilde{\mathtt{Hardswish}}$ loops over each component i and constrains $\widehat{X}_i$ to the linear piece closest to $X_i^{\mathtt{ref}}$. Recall that $\mathtt{Hardswish}(X)_i = \text{hardswish}(X_i)$ has two linear pieces: $\text{hardswish}(x) = x$ if $x \geq 3$ and $\text{hardswish}(x) = 0$ if $x \leq -3$.

S1a If $X_i^{\mathtt{ref}} \geq 0$, Line 42 constrains $\widehat{X}_i$ to the closest linear piece $\text{hardswish}(x) = x$ where $x \geq 3$ with $\widehat{X}_i \geq 3$, which implies $\widehat{Y}_i \overset{\text{def}}{=} \widehat{X}_i$. Here because $X_0^{\mathtt{ref}} = 5 \geq 0$, $\widehat{X}_0 \geq 3$ is in φ and $\widehat{Y}_0 \overset{\text{def}}{=} \widehat{X}_0$.

S1b If $X_i^{\mathtt{ref}} < 0$, Line 43 constrains $\widehat{X}_i$ to the closest linear piece $\text{hardswish}(x) = 0$ where $x \leq -3$ with $\widehat{X}_i \leq -3$, which implies $\widehat{Y}_i \overset{\text{def}}{=} 0$. Here because $X_1^{\mathtt{ref}} = -2 < 0$, $\widehat{X}_1 \leq -3$ is in φ and $\widehat{Y}_1 \overset{\text{def}}{=} 0$.

Next we present the conditional symbolic forward execution $\widetilde{\mathcal{N}}[X_{\mathtt{ref}}^{(0)}]$ *for a concrete point* $X^{(0)}$ (Line 17 in Algorithm 1) via a walk-through example. Let $\widehat{X}^{(L)}, \varphi \overset{\text{def}}{=} \widetilde{\mathcal{N}}[X_{\mathtt{ref}}^{(0)}](X^{(0)})$. $\widetilde{\mathcal{N}}[X_{\mathtt{ref}}^{(0)}]$ takes a concrete point $X^{(0)}$, makes the first layer weight $\widehat{W}^{(0)}$ and all layers' bias $\widehat{B}^{(\ell)}$ symbolic, then uses conditional symbolic activation functions to forward execute $X^{(0)}$ with the reference point $X_{\mathtt{ref}}^{(0)}$. It returns the symbolic output point $\widehat{X}^{(L)}$ with constraint φ.

Example 4.7. Consider the DNN $\mathcal{N}_1$ (Figure 5(a)) and input point $X^{(0)} \overset{\text{def}}{=} \begin{bmatrix}-1.5\end{bmatrix}$ from §3.1. Let $\widehat{X}^{(2)}, \varphi \overset{\text{def}}{=} \widetilde{\mathcal{N}_1}\left[X^{(0)}_{\texttt{ref}}\right]\left(X^{(0)}\right)$ where $X^{(0)}_{\texttt{ref}} \overset{\text{def}}{=} X^{(0)}$. We will show that $\widehat{X}^{(2)} = \begin{bmatrix}-0.75\widehat{W}^{(0)}_{0,0} + 0.5\widehat{B}^{(0)}_0 + \widehat{B}^{(1)}_0\end{bmatrix}$ and $\varphi = \varphi^{(1)}_\sigma$ where $\varphi^{(1)}_\sigma \overset{\text{def}}{=} -1.5\widehat{W}^{(0)}_{0,0} + \widehat{B}^{(0)}_0 \geq 0 \wedge -1.5\widehat{W}^{(0)}_{0,1} + \widehat{B}^{(0)}_1 < 0 \wedge -1.5\widehat{W}^{(0)}_{0,2} + \widehat{B}^{(0)}_2 < 0$.

Lines 24–29 runs $\mathcal{N}_1$ on $X^{(0)}$ layer by layer. The following steps are used for the first layer $\mathcal{N}^{(0)}_1$:

S1a Line 25 applies the layer affine transformation $\widehat{X}^{(1)}_{\texttt{pre}} \leftarrow X^{(0)}\widehat{W}^{(0)} + \widehat{B}^{(0)}$ with the symbolic layer weight $\widehat{W}^{(0)}$ and bias $\widehat{B}^{(0)}$ to the layer input $X^{(0)}$. $\widehat{X}^{(1)}_{\texttt{pre}} \overset{\text{def}}{=} \begin{bmatrix}-1.5\widehat{W}^{(0)}_{0,0} + \widehat{B}^{(0)}_0 & -1.5\widehat{W}^{(0)}_{0,1} + \widehat{B}^{(0)}_1 & -1.5\widehat{W}^{(0)}_{0,2} + \widehat{B}^{(0)}_2\end{bmatrix}$.

S2 Line 27 applies the layer affine transformation $X^{(1)}_{\texttt{ref}} \leftarrow X^{(0)}_{\texttt{ref}} W^{(0)} + B^{(0)}$ with the original layer weight $W^{(0)}$ and bias $B^{(0)}$ to the reference point $X^{(0)}_{\texttt{ref}}$, resulting in $X^{(1)}_{\texttt{ref}} \overset{\text{def}}{=} \begin{bmatrix}1.5 & -3.5 & -0.75\end{bmatrix}$.

S3 Line 28 applies the conditional activation function $\widehat{X}^{(1)}, \varphi^{(1)}_\sigma \leftarrow \widetilde{\sigma}^{(0)}\left[X^{(1)}_{\texttt{ref}}\right]\left(\widehat{X}^{(1)}_{\texttt{pre}}\right)$ with the transformed reference point $X^{(1)}_{\texttt{ref}}$ to $\widehat{X}^{(1)}_{\texttt{pre}}$, where $\widehat{X}^{(1)} \overset{\text{def}}{=} \begin{bmatrix}-1.5\widehat{W}^{(0)}_{0,0} + \widehat{B}^{(0)}_0 & 0 & 0\end{bmatrix}$ and $\varphi^{(1)}_\sigma$ is shown above.

S4 Line 29 applies the activation function $X^{(1)}_{\texttt{ref}} \leftarrow \sigma^{(0)}\left(X^{(1)}_{\texttt{ref}}\right)$ to $X^{(1)}_{\texttt{ref}}$, resulting in $X^{(1)}_{\texttt{ref}} \overset{\text{def}}{=} \begin{bmatrix}1.5 & 0 & 0\end{bmatrix}$.

For the second layer $\mathcal{N}^{(1)}_1$, a different Step **S1b** is used to avoid quadratic terms:

S1b Line 26 applies the layer affine transformation $\widehat{X}^{(2)}_{\texttt{pre}} \leftarrow \widehat{X}^{(1)} W^{(1)} + \widehat{B}^{(1)}$ with the original layer weight $W^{(1)}$ and symbolic bias $\widehat{B}^{(1)}$ to the symbolic layer input $\widehat{X}^{(1)}$. $\widehat{X}^{(2)}_{\texttt{pre}} \overset{\text{def}}{=} \begin{bmatrix}-0.75\widehat{W}^{(0)}_{0,0} + 0.5\widehat{B}^{(0)}_0 + \widehat{B}^{(1)}_0\end{bmatrix}$.

Then steps **S2**, **S3** and **S4** are repeated. Step **S2** results in $X^{(2)}_{\texttt{ref}} \overset{\text{def}}{=} \begin{bmatrix}0.25\end{bmatrix}$. Step **S3** results in $\widehat{X}^{(2)} \overset{\text{def}}{=} \widehat{X}^{(2)}_{\texttt{pre}}$ and $\varphi^{(2)}_\sigma \overset{\text{def}}{=} \top$ because $\mathcal{N}^{(1)}_1$ is the last layer with an identity activation function. Step **S4** results in $X^{(2)}_{\texttt{ref}} \overset{\text{def}}{=} \begin{bmatrix}0.25\end{bmatrix}$. Finally, on Line 30, $\widehat{X}^{(2)}$ and $\bigwedge_{1 \leq \ell \leq 2} \varphi^{(\ell)}_\sigma = \varphi^{(1)}_\sigma$ are returned.

Theorem 4.8. *Let $\widehat{X}^{(L)}, \varphi \overset{\text{def}}{=} \widetilde{\mathcal{N}}\left[X^{(0)}_{\texttt{ref}}\right]\left(X^{(0)}\right)$ and $\theta \vDash \varphi$, then*

C1 $[\![\widehat{X}^{(L)}]\!]_\theta = \mathcal{N}^\theta\left(X^{(0)}\right)$.

C2 *$\widehat{X}^{(L)}$ is a linear expression and φ is a linear formula.*

C3 *Let the V-polytope $P^{(0)} \overset{\text{def}}{=} \left\{{}_0X^{(0)}, {}_1X^{(0)}, \ldots, {}_{p-1}X^{(0)}\right\}$, ${}_j\widehat{X}^{(L)}, {}_j\varphi \overset{\text{def}}{=} \widetilde{\mathcal{N}}\left[X^{(0)}_{\texttt{ref}}\right]\left({}_jX^{(0)}\right)$ for $0 \leq j < p$, and $\varphi_P \overset{\text{def}}{=} \bigwedge_j {}_j\varphi$. If $\theta \vDash \varphi_P$, then $\mathcal{N}^\theta$ is locally linear for the polytope $P^{(0)}$.*

Proof Sketch. Condition **C1** states the correctness of the conditional encoding for $\widehat{X}^{(L)}$. By Definition 4.3-**C1**, for the first layer, we have $[\![\widehat{X}^{(1)}]\!]_\theta = \sigma^{(0)}\left([\![\widehat{X}^{(1)}_{\texttt{pre}}]\!]_\theta\right) = \sigma^{(0)}\left(X^{(0)}[\![\widehat{W}^{(0)}]\!]_\theta + [\![\widehat{B}^{(0)}]\!]_\theta\right) = \mathcal{N}^{(0)}\left(X^{(0)}; [\![\widehat{W}^{(0)}]\!]_\theta, [\![\widehat{B}^{(0)}]\!]_\theta\right)$. For later layers $\ell \geq 1$, $[\![\widehat{X}^{(\ell+1)}]\!]_\theta = \sigma^{(\ell)}\left([\![\widehat{X}^{(\ell+1)}_{\texttt{pre}}]\!]_\theta\right) = \sigma^{(\ell)}\left([\![\widehat{X}^{(\ell)}]\!]_\theta W^{(\ell)} + [\![\widehat{B}^{(\ell)}]\!]_\theta\right) = \mathcal{N}^{(\ell)}\left([\![\widehat{X}^{(\ell)}]\!]_\theta; W^{(\ell)}, [\![\widehat{B}^{(\ell)}]\!]_\theta\right)$. Thus, by induction, $[\![\widehat{X}^{(L)}]\!]_\theta = \mathcal{N}^\theta\left(X^{(0)}\right)$. Condition **C2** follows from Definition 4.3-**C2** and is necessary to formulate the repair problem as an LP problem. Condition **C3** is necessary for provable polytope repair. It lifts and can be proved using Definition 4.3-**C3**. □

Next we present the conditional symbolic forward execution $\widetilde{\mathcal{N}}\left(P^{(0)}\right)$ (Line 22 in Algorithm 1) that lifts $\widetilde{\mathcal{N}}\left[X^{(0)}_{\texttt{ref}}\right]\left(X^{(0)}\right)$ for a concrete point $X^{(0)}$ to a concrete V-polytope $P^{(0)}$.

Example 4.9. Consider the DNN $\mathcal{N}_1$ (Figure 5(a)) and polytope $P_1 \overset{\text{def}}{=} \left\{\begin{bmatrix}-1.5\end{bmatrix}, \begin{bmatrix}-0.5\end{bmatrix}\right\}$ from §3.2. Let $\widehat{P}^{(2)}, \varphi \overset{\text{def}}{=} \widetilde{\mathcal{N}_1}\left(P_1\right)$. We will show that φ is the first two rows of Equation 3, $\widehat{P}^{(2)} \overset{\text{def}}{=} \left\{{}_0\widehat{X}^{(2)}, {}_1\widehat{X}^{(2)}\right\}$ where ${}_0\widehat{X}^{(2)} \overset{\text{def}}{=} \begin{bmatrix}-0.75\widehat{W}^{(0)}_{0,0} + 0.5\widehat{B}^{(0)}_0 + \widehat{B}^{(1)}_0\end{bmatrix}$ and ${}_1\widehat{X}^{(2)} \overset{\text{def}}{=} \begin{bmatrix}-0.25\widehat{W}^{(0)}_{0,0} + 0.5\widehat{B}^{(0)}_0 + \widehat{B}^{(1)}_0\end{bmatrix}$.

Line 20 conditionally forwards each vertex ${}_j\widehat{X}^{(L)}, {}_j\varphi_\sigma \leftarrow \widetilde{\mathcal{N}}\left[\texttt{calc\_ref}\left(P^{(0)}\right)\right]\left({}_jX^{(0)}\right)$ with the same reference point $\texttt{calc\_ref}\left(P^{(0)}\right)$. `calc_ref` is a custom function that deterministically maps

$P^{(0)}$ to a concrete point. In this example we assume `calc_ref`$(P^{(0)}) = [-1.5]$. For ${}_0X^{(0)} \overset{\text{def}}{=} [-1.5]$, as presented in Example 4.7, ${}_0\widehat{\boldsymbol{X}}^{(L)} \overset{\text{def}}{=} \left[-0.75\widehat{\boldsymbol{W}}^{(0)}_{0,0}+0.5\widehat{\boldsymbol{B}}^{(0)}_0+\widehat{\boldsymbol{B}}^{(1)}_0\right]$ and ${}_0\varphi_\sigma$ is the first row of Equation 3. For ${}_1X^{(0)} \overset{\text{def}}{=} [-0.5]$ we can compute ${}_1\widehat{\boldsymbol{X}}^{(L)} \overset{\text{def}}{=} \left[-0.25\widehat{\boldsymbol{W}}^{(0)}_{0,0}+0.5\widehat{\boldsymbol{B}}^{(0)}_0+\widehat{\boldsymbol{B}}^{(1)}_0\right]$ in the same way, and ${}_1\varphi_\sigma$ is the second row of Equation 3. On Line 21, $\left\{{}_0\widehat{\boldsymbol{X}}^{(2)}, {}_1\widehat{\boldsymbol{X}}^{(2)}\right\}$ and ${}_0\varphi_\sigma \wedge {}_1\varphi_\sigma$ are returned.

Theorem 4.10. *Let the polytope* $P^{(0)} \overset{\text{def}}{=} \left\{{}_0X^{(0)}, {}_1X^{(0)}, \ldots, {}_{p-1}X^{(0)}\right\}$. *Let* $\widehat{\boldsymbol{P}}^{(L)}, \varphi \overset{\text{def}}{=} \widetilde{\mathcal{N}}(P^{(0)})$ *where* $\widehat{\boldsymbol{P}}^{(L)} = \left\{{}_0\widehat{\boldsymbol{X}}^{(L)}, {}_1\widehat{\boldsymbol{X}}^{(L)}, \ldots, {}_{p-1}\widehat{\boldsymbol{X}}^{(L)}\right\}$. *Let* $\theta \models \varphi$.

C1 $\mathcal{N}^\theta({}_jX^{(0)}) = [\![{}_j\widehat{\boldsymbol{X}}^{(L)}]\!]_\theta$ *for* $0 \le j < p$.
C2 ${}_j\widehat{\boldsymbol{X}}^{(L)} \in \widehat{\boldsymbol{P}}^{(L)}$ *is a linear expression and* φ *is a linear formula for* $0 \le j < p$.
C3 $\theta \models \varphi$ *implies that* $\mathcal{N}^\theta$ *is locally linear for* $P^{(0)}$.

Proof Sketch. The proof of this theorem follows from Theorem 4.8. □

The following example demonstrates `Shift&Assert` (Line 6 in Algorithm 1).

Example 4.11. Consider the DNN $\mathcal{N}_4$ (Figure 8(a)) and the set of V-polytopes $\mathcal{P}_2$ from §3.2. We will show $\mathcal{N}_5 =$ `Shift&Assert`$(\mathcal{N}_4, \mathcal{N}_4, \mathcal{P}_2, \Psi_2, k)$ where $\mathcal{N}_5$ is shown in Figure 8(c) and $k \overset{\text{def}}{=} 1$.

S1 Lines 8–10 conditionally forward executes each input V-polytope $P^{(k)} \in \mathcal{N}^{(0:k)}(\mathcal{P}^{(0)})$ on $\widetilde{\mathcal{N}}^{(k:L)}$. In this example, $\widehat{\boldsymbol{P}}^{(2)}, \varphi_\sigma \leftarrow \widetilde{\mathcal{N}_4}^{(1:2)}(P^{(1)})$ for each $P^{(1)} \in \mathcal{N}_4^{(0:1)}(\mathcal{P}_2)$.
S2 Line 11 adds the repair specification $\Psi(\widehat{\boldsymbol{\mathcal{P}}}^{(L)})$ to φ. In this example, $\Psi_2(\widehat{\boldsymbol{\mathcal{P}}}^{(L)})$ is added to φ.
S3 Lines 12–13 solves the LP problem. Line 12 constructs the delta vector $\vec{\Delta}$ that consists of both the functional difference $\widehat{\boldsymbol{\mathcal{P}}}^{(L)} - \mathcal{N}_{\text{og}}(\mathcal{P}^{(0)})$ and the parameter difference $\widehat{\boldsymbol{\theta}}_{\mathcal{N}^{(k:L)}} - \theta_{\mathcal{N}^{(k:L)}}$. Line 13 calls an LP solver to find new parameters $\theta'_{\mathcal{N}^{(k:L)}}$ that satisfy φ while minimizing $\|\vec{\Delta}\|_p$. In this example, $\mathcal{N} \overset{\text{def}}{=} \mathcal{N}_4$, $\mathcal{N}_{\text{og}} \overset{\text{def}}{=} \mathcal{N}_4$, $\mathcal{P}^{(0)} \overset{\text{def}}{=} \mathcal{P}_2$, $k \overset{\text{def}}{=} 1$ and $L \overset{\text{def}}{=} 2$.
S4 Line 14 returns $\perp$ if φ is unsatisfiable. Otherwise Line 15 updates $\mathcal{N}_{\texttt{ret}}^{(1:2)}$ with the new parameter $\theta'_{\mathcal{N}^{(k:L)}}$ where $\mathcal{N}_{\texttt{ret}}$ is a copy of $\mathcal{N} \overset{\text{def}}{=} \mathcal{N}_4$. Finally Line 16 returns the repaired $\mathcal{N}_5 \overset{\text{def}}{=} \mathcal{N}_{\texttt{ret}}$.

Theorem 4.12. *Let* $\mathcal{N}_{\texttt{ret}} \overset{\text{def}}{=}$ `Shift&Assert`$(\mathcal{N}, \mathcal{N}^{\text{og}}, \mathcal{P}^{(0)}, \Psi, k)$, $\mathcal{N}_{\texttt{ret}} \neq \perp$ *and* $\mathcal{N}^{(0:k)}$ *is locally linear for any polytope* $P^{(0)} \in \mathcal{P}^{(0)}$. *Then:*

C1 $\mathcal{N}_{\texttt{ret}}$ *is locally linear for any polytope* $P^{(0)} \in \mathcal{P}^{(0)}$.
C2 $\mathcal{N}_{\texttt{ret}}$ *satisfies the polytope specification* $(\mathcal{P}^{(0)}, \Psi)$.
C3 `Shift&Assert` *runs in polynomial time in the total number of vertices in* $\mathcal{P}^{(0)}$ *and the size of* $\mathcal{N}$.

Proof Sketch. Given $\mathcal{N}^{(0:k)}$ is locally linear for $\mathcal{P}^{(0)}$ and $\mathcal{N}_{\texttt{ret}}^{(0:k)}$ is the same as $\mathcal{N}^{(0:k)}$. Because $\mathcal{N}_{\texttt{ret}} \neq \perp$, $\theta'_{\mathcal{N}^{(k:L)}}$ satisfies φ. Hence $\mathcal{N}_{\texttt{ret}}^{(k:L)}$ with the new parameters $\theta'_{\mathcal{N}^{(k:L)}}$ is locally linear for any polytope $P^{(k)} \in \mathcal{N}_{\texttt{ret}}^{(0:k)}(\mathcal{P}^{(0)})$ using Theorem 4.10-**C1**. Thus, we have $\mathcal{N}_{\texttt{ret}}$ is locally linear for any polytope $P^{(0)} \in \mathcal{P}^{(0)}$, proving Condition **C1**.

Consider any polytope $P^{(0)} \in \mathcal{P}^{(0)}$ and its corresponding specification $\psi \in \Psi$, $\theta'_{\mathcal{N}^{(k:L)}} \models \varphi$ implies that $\mathcal{N}_{\texttt{ret}}$ satisfies ψ on all vertices of $P^{(0)}$. Because ψ is a linear formula and $\mathcal{N}_{\texttt{ret}}$ is locally linear for $P^{(0)}$, $\mathcal{N}_{\texttt{ret}}$ satisfies ψ on all points in `ConvexHull`$(P^{(0)})$. Thus, $\mathcal{N}_{\texttt{ret}}$ satisfies the polytope specification $(\mathcal{P}^{(0)}, \Psi)$, proving Condition **C2**.

Condition **C3** is true because the size of linear formula φ is polynomial in the total number of vertices in $\mathcal{P}^{(0)}$ and the size of $\mathcal{N}$. □

The following example demonstrates `VPolytopeRepair` defined in Algorithm 1.

Definition 4.13. Given a DNN $\mathcal{N}$ of L layers, a layer index k, and input V-polytopes $\mathcal{P}^{(0)}$. A network partition s, which is a list of network index tuples $[(k_0, l_0), (k_1, l_1), \ldots, (k_{n-1}, l_{n-1})]$, is valid for $\mathcal{N}$, k and $\mathcal{P}^{(0)}$ if either s is empty and $\mathcal{N}^{(0:k)}$ is locally linear for $\mathcal{P}^{(0)}$, or s satisfies: (a) for any tuple (k_i, l_i) in s, $0 \leq k_i < l_i \leq L$; (b) for the first tuple (k_0, l_0) in s, $\mathcal{N}^{(0:k_0)}$ is locally linear for $\mathcal{P}^{(0)}$; (c) for any two consecutive tuples (k_i, l_i) and (k_{i+1}, l_{i+1}) in s, $k_{i+1} \leq l_i$; (d) For the last tuple (k_{n-1}, l_{n-1}) in s, $k \leq l_{n-1}$.

Example 4.14. For the V-polytope repair in §3.2, $\mathcal{N}_5 = \texttt{VPolytopeRepair}\big(\mathcal{N}_1, \mathcal{P}_2^{(0)}, \Psi_2, s, k\big)$ where $s = [(0, 1)]$ and $k = 1$.

S1 Line 1 makes a copy $\mathcal{N}_{\texttt{ret}}$ of $\mathcal{N}$ for repair. Lines 2–4 shift $\mathcal{N}_{\texttt{ret}}$ such that $\mathcal{N}_{\texttt{ret}}^{(0:k)}$ is locally linear for $\mathcal{P}^{(0)}$. Specifically, for each (k_i, ℓ_i) pair in the network partition s, Line 3 requires $\mathcal{N}_{\texttt{ret}}^{(0:k_i)}$ to be locally linear for $\mathcal{P}^{(0)}$ and shifts $\mathcal{N}_{\texttt{ret}}^{(k_i:\ell_i)}$ with an empty specification $\top$ such that $\mathcal{N}_{\texttt{ret}}^{(0:\ell_i)}$ is locally linear for $\mathcal{P}^{(0)}$. The original $\mathcal{N}$ is used to minimize the functional difference. Line 4 returns $\bot$ if `Shift&Assert` fails. In this example, after the pair $(0, 1) \in s$, $\mathcal{N}_{\texttt{ret}}$ is $\mathcal{N}_4$ shown in Figure 8(a).
S2 Line 5 requires $\mathcal{N}_{\texttt{ret}}^{(0:k)}$ to be locally linear for $\mathcal{P}^{(0)}$, calls $\texttt{Shift\&Assert}\big(\mathcal{N}_{\texttt{ret}}, \mathcal{N}, \mathcal{P}^{(0)}, \Psi, k\big)$ to repair the shifted DNN $\mathcal{N}_{\texttt{ret}}$ against the V-polytope specification $(\mathcal{P}^{(0)}, \Psi)$ and returns its result. In this example, $\texttt{Shift\&Assert}\big(\mathcal{N}_4, \mathcal{N}_1, \mathcal{P}_2^{(0)}, \Psi_2, 1\big)$ returns $\mathcal{N}_5$ shown in Figure 8(c).

Theorem 4.15. *Let* $\mathcal{N}_{\texttt{ret}} \overset{def}{=} \texttt{VPolytopeRepair}\big(\mathcal{N}, \mathcal{P}^{(0)}, \Psi, s, k\big)$, $\mathcal{N}_{\texttt{ret}} \neq \bot$, *$s$ is a valid network partition for $\mathcal{N}$ and k.* `VPolytopeRepair` *satisfies the following properties:*

C1 *$\mathcal{N}_{\texttt{ret}}$ satisfies the V-polytope specification $\big(\mathcal{P}^{(0)}, \Psi\big)$.*
C2 `VPolytopeRepair` *runs in polynomial time in the number of vertices in $\mathcal{P}^{(0)}$, size of $\mathcal{N}$ and s.*

Proof. Because Step **S1** makes $\mathcal{N}_{\texttt{ret}}^{(0:k)}$ locally linear for $\mathcal{P}^{(0)}$ (using Theorem 4.12-**C1** and Definition 4.13), Step **S2** returns a DNN $\mathcal{N}_{\texttt{ret}}$ that satisfies the V-polytope repair specification if $\mathcal{N}_{\texttt{ret}} \neq \bot$ (using Theorem 4.12-**C2**), proving Condition **C1**. Condition **C2** follows from Theorem 4.12-**C3** as `VPolytopeRepair` calls `Shift&Assert` for each partition in s and once on Line 5. □

5 QUALITATIVE COMPARISON WITH PRIOR APPROACHES

In this section, we present a qualitative comparison between APRNN (this work) and prior provable-repair techniques, PRDNN [Sotoudeh and Thakur 2021b] and REASSURE [Fu and Li 2022].
Pointwise repair. PRDNN and REASSURE are not architecture-preserving for provable pointwise repair, unlike APRNN. PRDNN introduces the notion of a *Decoupled* DNN, which decouples the original DNN into an activation network and a value network. The approach then reduces repairing a single layer in the value network to solving an LP problem.

REASSURE reduces repairing a point X to repairing the linear region $\mathcal{A}$ containing that point. For a linear region $\mathcal{A}$, REASSURE adds a patch network $p_{\mathcal{A}}$ such that $\mathcal{N}(X) + p_{\mathcal{A}}(X)$ for $X \in \mathcal{A}$ satisfies the repair specification. To localize changes to $\mathcal{A}$ while keeping the repaired network $\mathcal{N}'$ continuous, REASSURE constructs a support network $g_{\mathcal{A}}$ such that $\mathcal{N}'(X)$ equals to $\mathcal{N}(X) + p_{\mathcal{A}}(X)$ for $X \in \mathcal{A}$ and smoothly goes to the original output $\mathcal{N}(X)$ for X outside $\mathcal{A}$.
Polytope repair. APRNN supports arbitrary V-polytope repair specifications. PRDNN and REASSURE need to enumerate linear regions, unlike APRNN. To ensure scalability, PRDNN only supports V-polytopes in 2D subspaces of the input space. It uses SyReNN [Sotoudeh et al. 2023; Sotoudeh and Thakur 2021c] to efficiently enumerate the linear regions, and then performs pointwise repair on the vertices of these linear regions to guarantee provable V-polytope repair.

REASSURE reduces the repair for a polytope to the repair for all buggy linear regions in the polytope. However, REASSURE cannot be used in practice for V-polytope or H-polytope repair because a polytope has a tremendous amount of linear regions, which grows exponentially in the network depth. Even just enumerating and finding all buggy linear regions is impractical, and the REASSURE tool does not provide a way to do so. Moreover, even if one could provide REASSURE with all buggy linear regions, the repaired network produced by REASSURE has large runtime and memory overhead (as demonstrated later in this section and §7.8).

Soundness. Both APRNN and PRDNN are sound; that is, if a repaired DNN is generated by APRNN or PRDNN, then it is guaranteed to satisfy the repair specification.

REASSURE is unsound for pointwise repair when two points in the same linear region have conflicting specifications: REASSURE is unable to identify this conflict, and the tool will return an incorrect DNN instead of reporting that the repair is infeasible for REASSURE.

REASSURE is unsound for polytope repair in at least two cases. Consider two input polytopes with different specifications. For any buggy linear that intersects with both input polytopes, REASSURE will generate an incorrect repair by adding two overlapping patch networks of the two specifications separately for such linear region. Moreover, for two adjacent buggy linear regions with different specifications, the behavior on their common boundary may violate both specifications due to the support networks.

Completeness for pointwise repair. APRNN, PRDNN and REASSURE are complete for single-point repair; that is, they are guaranteed to generate a repaired DNN that satisfies the single-point repair specification. However, they are incomplete for general pointwise repair. Given a DNN $\mathcal{N}$, consider a pointwise repair specification with three points in the same linear region of $\mathcal{N}$, but with corresponding outputs constrained to be non-collinear; e.g., DNN $\mathcal{N}_1$ in Figure 5(a) and the pointwise specification $\mathcal{N}_1\big(\big[2.5\big]\big)_0 \leq 0.4 \wedge \mathcal{N}_1\big(\big[3\big]\big)_0 \geq 0.5 \wedge \mathcal{N}_1\big(\big[3.5\big]\big)_0 \leq 0.5$. Although a repair exists, PRDNN fails because the decoupled activation network keeps buggy points in their original linear region. REASSURE also fails because it is impossible to repair the same linear region for three buggy points with conflicting constraints. The REASSURE implementation fails to detect such a conflict and gives an incorrect repair. APRNN will also fail if the reference points keep the three points in the same linear region. However, APRNN provides the flexibility to try different reference points such that not all three points are in the same linear region in the repaired DNN. For example, with reference points $X^{(0)}_{\mathtt{ref}} = \begin{bmatrix}1.5 & 3 & 3.5\end{bmatrix}$, APRNN finds a repair.

Drawdown and generalization. APRNN and PRDNN both minimize changes in the parameters and functional difference as a proxy for minimizing drawdown. REASSURE guarantees that changes are localized to the repaired linear regions. In practice, APRNN and PRDNN have much higher generalization with REASSURE exhibiting 0% generalization (see §7.1).

Activation function. APRNN supports activation functions that have linear pieces. PRDNN theoretically supports any activation function for pointwise repair and piecewise-linear functions for polytope repair, but the implementation only supports piecewise-linear functions. REASSURE only supports piecewise-linear functions.

Network-size overhead. Being architecture preserving, APRNN does not have any network size overhead. Because PRDNN converts the given DNN into a Decoupled DNN, the repaired network size doubles in a naive implementation. An optimized implementation could only store the original DNN and the single repaired value layer. The network-size overhead for PRDNN does not depend on the given repair specification.

In contrast, REASSURE has a significant overhead in network size even when repairing a single linear region, and the network-size overhead depends on the number of linear regions being repaired. Consider a fully-connected ReLU network of L layers. To repair one linear region, REASSURE adds a patch network p, a support network g and a scalar parameter of the $\|p(X)\|_\infty$. A patch network consists of a $n^{(0)} \times n^{(L)}$ fully-connected layer with $n^{(0)} \times n^{(L)} + n^{(L)}$ parameters. A support network consists of a $n^{(0)} \times n^{(hidden)}$ fully-connected layer with $n^{(0)} \times n^{(hidden)} + n^{(hidden)}$ parameters where $n^{(hidden)} \overset{\text{def}}{=} \sum_{\ell=1}^{L-1} n^{(\ell)}$ denotes the amount of all hidden neurons. Therefore, *for each linear region*, the REASSURE repaired network adds $(n^{(0)} + 1) \times (n^{(L)} + \sum_{\ell=1}^{L-1} n^{(\ell)}) + 1$ new parameters. For example, the MNIST 3×100 network from §7.1 has 89,610 parameter, and REASSURE will add 164,851 *new* parameters *for each buggy linear region.*

6 IMPLEMENTATION

APRNN is built using PyTorch [Paszke et al. 2019], an open source machine learning framework, and Gurobi [Gurobi Optimization, LLC 2022], a mathematical optimization solver for linear programming (LP), quadratic programming (QP) and mixed integer programming (MIP) problems. The code is available at https://github.com/95616ARG/APRNN.

To measure parity with APRNN, we also re-implemented PRDNN [Sotoudeh and Thakur 2021b] in our tool. Our tool provides an easier interface with which to repair a network using PRDNN, and allows for a fair comparison between the two algorithms by ensuring that the same repair specifications and networks are used. Compared to the original implementation of PRDNN [Sotoudeh and Thakur 2021a], the use of PyTorch decreases the time spent on the Jacobian computation. Further optimizations were achieved by making our PRDNN implementation GPU compatible.

We also implemented lookup-based override approaches for pointwise and V-polytope repair of classification DNNs. (a) For pointwise repair, we implemented a lookup-table approach that uses a hash-table of pointwise repair specification to override the output of the DNN. (b) For the V-polytope repair, we implemented a lookup-function approach that uses an LP solver to find inputs in the V-polytope repair specification and overrides output of the DNN for such inputs.

7 EXPERIMENTAL EVALUATION

All experiments were run on a machine with dual 16-core Intel Xeon Silver 4216 CPUs, 384 GB of memory, SSD and a NVIDIA RTX A6000 with 48 GB of GPU memory.

7.1 Pointwise MNIST Image Corruption Repair

Buggy networks. We repair MNIST fully-connected DNNs with ReLU activation layers from [Singh 2019] with a varying number of layers and parameters for classifying handwritten digits. We refer to the MNIST networks by the number of layers and the width of hidden layers. For example, the "3×100" network has 3 layers with 100 neurons per hidden layer ($784 \times 100 \times 100 \times 10$).
Pointwise repair specification. The MNIST-C dataset [Mu and Gilmer 2019] consists of images from the official MNIST test set that have been corrupted. The pointwise repair specification consists of the first 100 fog-corrupted images from MNIST-C. Buggy networks have 5%-15% accuracy on it.
Generalization set. 9,000 images from MNIST-C's foggy test set, disjoint from the repair set. The accuracy of the buggy DNNs on this generalization set is under 20%.
Drawdown set. The drawdown set is the official MNIST test set containing 10,000 images.
Results. All tools were able to repair all networks so they satisfied the pointwise repair specification. Table 1 summarizes the results. Both REASSURE and lookup-table have zero drawdown but zero generalization, while both APRNN and PRDNN have some drawdown but good generalization. APRNN has good (low) drawdown and the best (highest) generalization on all networks. PRDNN also has good generalization, but the drawdown is worse (higher) than APRNN.

Regarding repair time, the lookup-table finished instantly because it just caches the repair specification for lookup during inference. Both PRDNN and APRNN took a short amount of time,

Table 1. Provable pointwise repair of MNIST networks using APRNN (this work), PRDNN, REASSURE and lookup-table (LT). k: the pointwise repair parameter for PRDNN and APRNN, D: drawdown, G: generalization, T: repair time. APRNN uses empty $s = []$ for all repairs.

Network	APRNN				PRDNN				REASSURE			LT		
	k	D	G	T	k	D	G	T	D	G	T	D	G	T
3×100	1	1.28%	**31.53%**	5s	0	2.13%	23.34%	187s	**0%**	0%	490s	**0%**	0%	**<1s**
9×100	5	2.72%	**19.13%**	92s	5	9.55%	9.18%	11s	**0%**	0%	1503s	**0%**	0%	**<1s**
9×200	6	1.38%	**24.69%**	372s	1	3.92%	6.55%	15s	**0%**	0%	15356s	**0%**	0%	**<1s**

while PRDNN was faster than APRNN for two of the networks. By contrast, REASSURE took the longest time for all networks, and the time increases significantly as the network size increases. For the 9×200 network, REASSURE took 4 hours and 16 minutes. Moreover, the REASSURE repaired networks have significant runtime overhead, which is evaluated in §7.8.

This experiment demonstrates that APRNN is overall the best pointwise repair approach while preserving the original DNN architecture, because APRNN can efficiently repair a buggy network while maintaining good drawdown and achieving the best generalization.

7.2 Pointwise ImageNet Natural Adversarial Examples and Corruption Repair

Buggy networks. We repair modern ImageNet CNNs ResNet152 [He et al. 2016] (60.2 million parameters) and VGG19 [Simonyan and Zisserman 2015] (143.7 million parameters). ResNet152 has 78.312% top-1 accuracy and 94.046% top-5 accuracy. VGG19 has 72.376% top-1 accuracy and 90.876% top-5 accuracy.

Pointwise repair specification. We consider two pointwise repair specifications from the following two datasets: 1) the Natural Adversarial Examples (NAE) dataset [Hendrycks et al. 2019] that consists of 7,500 images which are commonly misclassified by modern ImageNet networks; 2) the ImageNet-C dataset [Hendrycks and Dietterich 2019] that consists of algorithmically generated corruptions applied to the ImageNet validation set with severity 1 to 5. For each network, 1) the pointwise repair specifications for the NAE dataset consist of the first 50 NAE images that were misclassified by that network; 2) the pointwise repair specifications for the ImageNet-C dataset consists of the first 50 fog-corrupted images of severity 3 that were misclassified by that network.

Generalization set. There is no generalization set for the NAE dataset because the images do not have common features to generalize. For the ImageNet-C dataset, it is reasonable to expect that the repair of one corrupted image generalizes to the same image with other severities of the same corruption. Thus, we take the same subset of fog-corrupted images as the corresponding repair specification but with different severities 1, 2, 4 and 5 as the generalization set.

Drawdown set. We use the ILSVRC2012 ImageNet validation set of 50,000 images [Deng et al. 2012] as the drawdown set. It is the standard dataset for evaluating the accuracy of ImageNet networks.

Repair layer. Both APRNN and PRDNN repair a convolutional layer in the last bottleneck block of ResNet152 and a fully-connected layer in the last classifier block of VGG19.

Results. For both NAE and ImageNet-C pointwise repair specifications, APRNN successfully repairs both ResNet152 and VGG19, while PRDNN runs out of memory during repair. For NAE, APRNN took 2,935 seconds to repair ResNet152 and 3,671 seconds to repair VGG19. The repaired ResNet152 has 1.33% top-1 and 0.66% top-5 drawdown, and the repaired VGG19 has 0.72% top-1 and 0.33% top-5 drawdown. For ImageNet-C, APRNN took 2,666 seconds to repair ResNet152 and 1,918 seconds to repair VGG19. The repaired ResNet152 has 1.32% top-1 and 0.56% top-5 drawdown, as well as 19.00% top-1 and 11.50% top-5 generalization; the repaired VGG19 has 0.02% top-1 and 0.00%

top-5 drawdown, as well as 23.00% top-1 and 20.00% top-5 generalization. This experiment highlights APRNN's ability to scale to very large networks with low drawdown and high generalization.

7.3 V-Polytope MNIST Image Corruption And Rotation Repair

Buggy network. In this experiment, we repair the MNIST 9×100 DNN from [Singh 2019].
V-polytope repair specification for APRNN and lookup-function. The V-polytope specification we use consists of n V-polytopes defined by m images along with the constraint that all images in the convex hull of this polytope have the same (correct) classification. Specifically, we take the first five foggy images with label "8" from the MNIST-C dataset that are misclassified by the network. We rotate each image $-30°$, $-20°$, $-10°$, $0°$, $10°$, $20°$, $30°$ in pixel space. The V-polytope repair specification consists of the five 784-dimensional V-polytopes of the convex hull of these seven rotated images with constraint that all images within each V-polytope are classified as "8".
Approximate 2D V-polytope repair specification for PRDNN and REASSURE. As discussed in §5, both PRDNN and REASSURE do not scale to higher-dimensional V-polytopes. Thus, we sample 2D linear regions from the true V-polytope repair specification to form an approximate V-polytope repair specification for PRDNN and REASSURE. We use SyReNN to sample 3,187 2D linear regions (of 12,394 unique vertices) from the 2D triangles among vertices of each V-polytope.
Approximate pointwise repair specification for REASSURE. Because REASSURE does not scale to repair a large number of linear regions (as discussed in §5 and shown in §7.1), we sample a smaller pointwise repair specification for it. This repair specification consists of 500 unique buggy linear regions sampled from the V-polytope repair specification, represented by 500 points in those linear regions. As a comparison, we also evaluate APRNN pointwise repair on this specification.
Generalization sets. We consider two generalization sets: (a) the generalization set S_1 consists of foggy misclassified images of label "8" (disjoint from those included in the repair set) rotated using the same seven rotations used to construct the repair specification; (b) the generalization set S_2 consists of the same five foggy misclassified images of label "8" that were chosen to form the repair specification, but rotated by angles ranging from $-30°$ to $30°$ with a step of $1°$. The buggy network has 36.53% accuracy on S_1 and 35.77% accuracy on S_2.
Drawdown set. The drawdown set is the official MNIST test set containing 10,000 images.
Results. For the (true) V-polytope repair specification, only APRNN and the lookup-function can provably repair it. APRNN took 59 seconds with parameters $s = [(0, 2), (2, 3), \ldots, (8, 9)]$ and $k = 8$, while the lookup-function finishes instantly because it just caches the specification for lookup during inference. However, APRNN has better drawdown and significantly better generalization compared to the lookup-function. APRNN has -0.07% drawdown, which indicates improvement in drawdown set accuracy, as well as 60.98% generalization on S_1 and 63.53% generalization on S_2. By contrast, the lookup-function has no drawdown and no generalization. Moreover, the lookup-function introduces significant runtime overhead to the network, which is evaluated in §7.8.

For the approximate 2D V-polytope repair specification, PRDNN took 3,834 seconds with parameter $k = 5$, while REASSURE *times out in one day*. PRDNN has -0.07% drawdown, 53.97% generalization on S_1 and 61.5% generalization on S_2. Although PRDNN provably repairs this approximate 2D V-polytope repair specification, it does not guarantee provable repair for the true V-polytope repair specification—there are still violations of the true specification.

For the approximate pointwise repair specification, APRNN took 88 seconds with parameters $s = []$ and $k = 7$, while REASSURE took 9,958 seconds. APRNN has no drawdown, 48.85% generalization on S_1 and 52.31% generalization on S_2. By contrast, REASSURE has no drawdown but also no generalization. Although both APRNN and REASSURE's repairs for this approximate specification do not guarantee provable repair for the true V-polytope repair specification, APRNN achieves much higher generalization in a significantly shorter time. Moreover, the REASSURE repaired network has a significant runtime overhead, which is evaluated in §7.8. The REASSURE repaired 9×100 network has $2,117\times$ more parameters (150,210 to 318,075,710); it takes one minute and 8 GB of GPU memory to evaluate one image using NVIDIA RTX A6000.

This experiment highlights APRNN's superior efficiency in provable V-polytope repair with low drawdown and high generalization.

7.4 V-Polytope ACAS Xu Local Safety and Robustness Repair

Buggy network. We repair ACAS Xu network $\mathcal{N}_{2,9}$ [Julian et al. 2018], which processes a five-dimensional input representing the state of an unmanned aircraft and an intruder, and issues navigation advisories to avoid the intruder. This network has seven layers and $13,350$ parameters.
V-polytope repair specification. There are five safety properties that the ACAS Xu network $\mathcal{N}_{2,9}$ should satisfy; denoted as ϕ_1, ϕ_2, ϕ_3, ϕ_4, and ϕ_8 in [Katz et al. 2017]. Let $\phi \overset{\text{def}}{=} \phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \wedge \phi_8$. There are known violations of safety property ϕ in network $\mathcal{N}_{2,9}$'s behavior [Katz et al. 2017]. We evenly partition the 5D input polytopes into boxes with a spacing of 0.05, and take 24 boxes that contain points violating safety property ϕ to form the V-polytope repair specification. For each 5D box P, the repair specification states that (a) for all points $X \in \texttt{ConvexHull}(P)$, the network $\mathcal{N}_{2,9}$ should satisfy ϕ, *and* (b) for all X and X' in $\texttt{ConvexHull}(P)$, $\operatorname{argmin}(\mathcal{N}_{2,9}(X)) = \operatorname{argmin}(\mathcal{N}_{2,9}(X'))$. Part (b) of the repair specification states that the repaired network should be robust in the polytope P. Note that ACAS Xu uses argmin instead of argmax to compute the output classification.
Approximate pointwise repair specification for PRDNN. Because PRDNN is unable to handle 5D input V-polytopes, we evenly sample each 5D box in the V-polytope repair specification with a spacing of 0.016 to form an approximate pointwise repair specification of 1,368 points. For each of these sampled points, the network has to satisfy the safety property ϕ. Further, to approximate robustness, all points sampled from the same 5D box has to have the same output classification.
Generalization set. The generalization set consists of 169,625 points that violate safety property ϕ, disjoint from the polytopes in the repair specification. We define *property generalization* as the percentage of points that satisfy ϕ in the repaired network; higher property generalization is better.
Drawdown set. The drawdown set consists of 283,772,487 points for which the original network $\mathcal{N}_{2,9}$ satisfies the safety property ϕ. We define *property drawdown* as the percentage of points in the drawdown set that do not satisfy ϕ in the repaired network; lower property drawdown is better.
Results. APRNN took 56 seconds to repair the network $\mathcal{N}_{2,9}$ to satisfy the V-polytope repair specification with parameters $s = [(0,1),(1,2),\dots,(5,6)]$ and $k = 6$. By contrast, PRDNN took 848 seconds to repair the network $\mathcal{N}_{2,9}$ to satisfy the approximate pointwise repair specification (with parameter $k = 6$), while the resulting network still violates the V-polytope repair specification. The property drawdown for APRNN is 0.60%, while that for PRDNN is 0.45%. The property generalization for APRNN is 100%, while that for PRDNN is 92.58%. This experiment shows APRNN's support for higher-dimensional V-polytope repair specifications and its superior efficiency compared to PRDNN.

7.5 V-Polytope ACAS Xu Global Safety Repair

Buggy network. We use the same ACAS Xu network $\mathcal{N}_{2,9}$ used in §7.4.
V-polytope repair specification. The V-polytope repair specification states that the safety property ϕ (from §7.4) should be satisfied by all points in the (valid) input space for the network $\mathcal{N}_{2,9}$. Specifically, we evenly partition the input polytopes for the five safety properties that are applicable to $\mathcal{N}_{2,9}$ into disjoint 5D boxes of size 0.13. Each of the 5D box is further partitioned into 120 disjoint 5D simplices among its 32 box vertices. The final set of 27,600 5D simplices covering the *entire valid input space* is used to define the V-polytope repair specification.

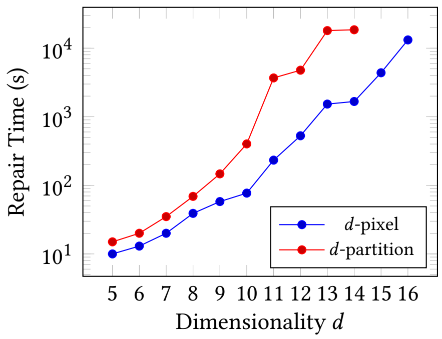


Fig. 9. APRNN repair time for MNIST d-dimensional L^∞ local-robustness repair.

Table 2. Provable pointwise repair of MNIST networks with non-PWL activation functions using APRNN. k: repair parameter, D: drawdown, G: generalization, T: time. The repair parameter s is empty for all runs.

Points	Hardswish				GELU			
	k	D	G	T	k	D	G	T
1	1	0.32%	5.54%	1s	1	0.05%	1.38%	1s
10	1	9.15%	13.08%	1s	1	13.24%	6.49%	1s
50	1	25.61%	17.53%	2s	1	32.60%	8.66%	3s
100	1	25.28%	25.44%	5s	1	42.21%	13.80%	8s

Results. APRNN took 102 seconds to repair the network $\mathcal{N}_{2,9}$ to satisfy the V-polytope repair specification with parameter $s = [(0,3), (3,4), (4,5), (5,6)]$ and $k = 6$. Because this repair covers the *entire* input space for each safety property, the resulting repaired DNN is guaranteed to satisfy *all* five safety properties. In other words, the repaired ACAS Xu $\mathcal{N}_{2,9}$ is provably safe. This experiment highlights APRNN's scalability to large numbers of V-polytopes in the repair specification and its ability to repair the entirety of an input space.

7.6 Pointwise MNIST Image Corruption Repair for Non-PWL DNNs

In this experiment we study APRNN's support for provably repairing Non-PWL DNNs.

Buggy networks. We repair MNIST 3×100 DNNs with only Hardswish or GELU activation functions. Hardswish is used in MobileNetV3 [Howard et al. 2019], and GELU is used in most transformer-based models like GPT-3 [Brown et al. 2020] and BERT [Devlin et al. 2019].

Pointwise repair specification. The MNIST-C dataset [Mu and Gilmer 2019] consists of MNIST images from the official MNIST test set that have been corrupted. The pointwise repair specifications consists of the first 1, 10, 50 and 100 fog-corrupted images from the MNIST-C dataset that were *misclassified* by that network along with their correct classification.

Generalization set. 9,000 images from MNIST-C's foggy test set, disjoint from the repair set.

Drawdown set. The drawdown set is the official MNIST test set containing 10,000 images.

Results. APRNN was able to repair all networks such that they satisfied the pointwise repair specification. Table 2 summarizes the results. As the number of repair points increases, the generalization improves (increases), but the drawdown worsens (increases). Although APRNN is able to provably repair DNNs with non-PWL activation functions, maintaining a good (low) drawdown down remains a challenge. We leave this as interesting future work.

7.7 d-Dimensional L^∞ Local-Robustness Repair for MNIST

In this experiment, we study the scalability of APRNN's provable V-polytope repair. For MNIST DNNs, an L^∞ local-robustness specification is expressed with a 784-dimensional cube of 2^{784} vertices in its V-representation. Such a V-polytope repair specification is challenging because APRNN's runs in polynomial time in the amount of vertices. We define two d-dimensional local-robustness specifications for MNIST where $d \leq 784$, and evaluate APRNN's scalability as d increases.

Buggy network. We repair the MNIST 3×100 DNN from [Singh 2019].

V-polytope repair specification. For each misclassified fog-corrupted image X of label l among five random choices from the MNIST-C dataset [Mu and Gilmer 2019], we consider the following two classes of d-dimensional local-robustness specifications where $5 \leq d \leq 16$ and $\varepsilon \overset{\text{def}}{=} 0.1$. (a) For a subset of d pixels (d-pixel), the DNN is locally robust to any L^∞-norm-bounded ε-perturbation of X on only those d pixels. Specifically, we consider the first d non-zero pixels and four random choices of d pixels. (b) For d equal-or-near-equal-sized partitions of all pixels (d-partition), the DNN is locally robust to any L^∞-norm-bounded ε-perturbation of X on those d partitions, where pixels in the same partition are perturbed with the same delta. Specifically we consider three partitions that consist of 4×4, 7×7 or 1×28 blocks.

Drawdown set. The drawdown set is the official MNIST test set containing 10,000 images.

Results. Figure 9 shows the average repair time for d-pixel and d-partition L^∞ local-robustness repairs with parameters $s = [(0, 1)]$ and $k = 1$ as well as a time limit of 20,000 seconds. The choice of images and the d pixels/partitions makes little difference in repair time and drawdown. For the d-pixel L^∞ local-robustness repair, APRNN scales to 16 dimensions (65,536 vertices) in 13,193 seconds. The worst drawdown is under 0.5% but most are under 0.2%. For the d-partition L^∞ local-robustness repair, APRNN scales to 14 dimensions (16,384 vertices) in 18,552 seconds. The worst drawdown is under 1% but most are under 0.2%.

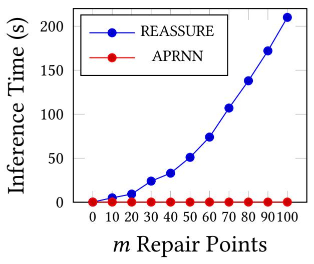
7.8 Runtime Overhead Analysis

In this experiment we evaluate the runtime overhead of non-architecture-preserving provable repair of DNNs by comparing APRNN with PRDNN, REASSURE and the lookup-based approach on pointwise and polytope repair. We use the MNIST 3×100 DNN from [Singh 2019].

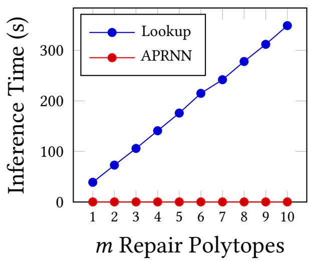
Results. We measure the inference time for 10,000 images from the MNIST test set in a single batch using GPU. The inference time for the original network is 0.2 seconds. APRNN is architecture-preserving, hence has no runtime overhead; the inference time of the APRNN repaired network is also 0.2 seconds. PRDNN's runtime overhead only depends on the network size, the inference time of the PRDNN repaired network is 0.35 seconds.

REASSURE has a *significant* runtime overhead, which is polynomial in the network size and the amount of buggy linear regions. For pointwise repair, Figure 10(a) shows the runtime overhead of REASSURE repaired networks for m points (which reduces to m linear regions). For polytope repair, REASSURE needs to enumerate and repair all buggy linear regions in each polytope. As described in §5, there can be a very large number of such linear regions and REASSURE does not provide a way to enumerate them. Thus, if the REASSURE is able to repair the network, then the runtime overhead will be significantly higher than the pointwise repair case.

For the lookup-based override approach, the runtime overhead for pointwise repair is constant due to the use of a hash-table; the inference time is 0.26 seconds. However, the runtime overhead for polytope repair is *significant* due to the use of LP solvers to check whether an input point is in a V-polytope. It is polynomial in the total amount of vertices of buggy V-polytopes. Figure 10(b) shows the runtime overhead of lookup-based override approach repaired networks for m 5D boxes (32 vertices per box).



(a) For pointwise repair.



(b) For polytope repair.

Fig. 10. Runtime overhead.

8 RELATED WORK

Provable DNN repair. Our work falls into the class of provable DNN repair methods. The most closely related works are PRDNN [Sotoudeh and Thakur 2021b], REASSURE [Fu and Li 2022] (discussed in §5), and Minimal Modifications of DNNs (MMDNN) [Goldberger et al. 2020]. MMDNN performs architecture-preserving provable pointwise repair of a DNN by reducing the problem to the NP-complete DNN verification problem. It is not able to scale to large DNNs, only supports the ReLU activation function, and does not support polytope repair or multi-layer repairs.

Heuristic DNN repair. Heuristic DNN repair methods do not provide guarantees about repair efficacy. MEND [Mitchell et al. 2022a] preserves the network architecture by training auxiliary editor networks to determine edits to the original DNN's parameters. MEND supports multi-layer edits, but does not handle V-polytope repair specifications.

SERAC [Mitchell et al. 2022b] does not preserve the network architecture; instead of altering the original buggy DNN's parameters, SERAC trains editor networks and modulates the final output based on the cached repair points and editor networks. SERAC supports the repair of DNNs with any activation functions, but does not handle V-polytope repair specifications.

NNRepair [Usman et al. 2021] modifies network parameters such that a given repair input follows a certain activation pattern. It produces a collection of experts (one per class label). NNRepair proposes three variants on how to combine these experts, and at least one of these is not architecture preserving. It does not support V-polytope repair specifications.

DNN training for repair. Training-based DNN repair methods support any activation functions and multi-layer edits, but do not provide any correctness guarantees. Sinitsin et al. [Sinitsin et al. 2020] propose editable neural networks, a framework that aims to make DNNs more amenable to an editing function that enforces changes in their behavior. Lin et al. [Lin et al. 2020] propose a method to use safety properties as loss functions during training. There are also certified training approaches for local [Mirman et al. 2018] and global robustness properties [Leino et al. 2021].

DNN verification. DNN verification methods that find buggy inputs provide motivation for provable DNN repair. Such DNN verification methods [Ferrari et al. 2022; Katz et al. 2017; Lahav and Katz 2021; Müller et al. 2022; Palma et al. 2021; Singh et al. 2018, 2019; Wang et al. 2018; Xu et al. 2021] can be used in conjunction with DNN repair to identify incorrect DNN behavior and verify the correctness of the DNN post-repair. Because APRNN is architecture-preserving, it can be easily integrated with these existing DNN verification methods.

9 CONCLUSION

We presented a new approach for architecture-preserving provable V-polytope repair of DNNs. Our algorithm runs in polynomial time. It supports a wide variety of activation functions and has the flexibility to modify weights in multiple layers. To the best of our knowledge, it is the first approach for provable DNN repair that supports all of these features. We implemented our approach in a tool called APRNN. Using MNIST, ImageNet, and ACAS Xu DNNs, our experiments showed that APRNN has better efficiency, scalability, and generalization compared to PRDNN and REASSURE, prior provable-repair techniques. We used V-polytope repair specifications to repair MNIST networks for rotations and ACAS Xu networks for robustness. We also showed how APRNN can be used to repair polytopes covering the entire input space of an ACAS Xu network; in effect, ensuring that the repaired network satisfies global safe properties. However, it is not feasible to represent an L^∞ local robustness specification on MNIST DNNs using V-polytopes due to the exponential number of vertices. Developing a provable repair approach that supports H-representation of polytopes is a natural and challenging direction for future research.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers as well as our shepherd Gagandeep Singh for their feedback and suggestions, which greatly improved the quality of the paper. This work is supported in part by NSF grant CCF-2048123 and DOE Award DE-SC0022285.

REFERENCES

David Bremner, Komei Fukuda, and Ambros Marzetta. 1997. Primal-dual methods for vertex and facet enumeration (preliminary version). In *Proceedings of the thirteenth annual symposium on Computational geometry.* 49–56.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

Jia Deng, Alex Berg, Sanjeev Satheesh, Hao Su, Aditya Khosla, and Fei-Fei Li. 2012. Imagenet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012). https://www.image-net.org/challenges/LSVRC/2012/

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021.* OpenReview.net. https://openreview.net/forum?id=YicbFdNTTy

Claudio Ferrari, Mark Niklas Müller, Nikola Jovanovic, and Martin T. Vechev. 2022. Complete Verification via Multi-Neuron Relaxation Guided Branch-and-Bound. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022.* OpenReview.net. https://openreview.net/forum?id=l_amHf1oaK

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

Feisi Fu and Wenchao Li. 2023. REASSURE. https://github.com/BU-DEPEND-Lab/REASSURE/tree/4781c8e781913598632da922fc8e41561f85e854/ICLR.

Ben Goldberger, Guy Katz, Yossi Adi, and Joseph Keshet. 2020. Minimal Modifications of Deep Neural Networks using Verification. In *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020 (EPiC Series in Computing, Vol. 73)*, Elvira Albert and Laura Kovács (Eds.). EasyChair, 260–278. https://doi.org/10.29007/699q

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning.* MIT Press. http://www.deeplearningbook.org.

Gurobi Optimization, LLC. 2022. Gurobi Optimizer Reference Manual. https://www.gurobi.com

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016.* IEEE Computer Society, 770–778. https://doi.org/10.1109/CVPR.2016.90

Dan Hendrycks and Thomas G. Dietterich. 2019. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019.* OpenReview.net. https://openreview.net/forum?id=HJz6tiCqYm

Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. 2019. Natural Adversarial Examples. *CoRR* abs/1907.07174 (2019). arXiv:1907.07174 http://arxiv.org/abs/1907.07174

Andrew Howard, Ruoming Pang, Hartwig Adam, Quoc V. Le, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, and Yukun Zhu. 2019. Searching for MobileNetV3. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019.* IEEE, 1314–1324. https://doi.org/10.1109/ICCV.2019.00140

Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *CoRR* abs/1602.07360 (2016). arXiv:1602.07360 http://arxiv.org/abs/1602.07360

Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. 2018. Deep Neural Network Compression for Aircraft Collision Avoidance Systems. *CoRR* abs/1810.04240 (2018). arXiv:1810.04240 http://arxiv.org/abs/1810.04240

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 *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10426)*, Rupak Majumdar and Viktor Kuncak (Eds.). Springer, 97–117. https://doi.org/10.1007/978-3-319-63387-9_5

Ronald Kemker, Marc McClure, Angelina Abitino, Tyler L. Hayes, and Christopher Kanan. 2018. Measuring Catastrophic Forgetting in Neural Networks. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, Sheila A. McIlraith and Kilian Q. Weinberger (Eds.). AAAI Press, 3390–3398. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16410

Leonid Genrikhovich Khachiyan. 1979. A polynomial algorithm in linear programming. In *Doklady Akademii Nauk*, Vol. 244. Russian Academy of Sciences.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger (Eds.). 1106–1114. https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

Ori Lahav and Guy Katz. 2021. Pruning and Slicing Neural Networks using Formal Verification. In *Formal Methods in Computer Aided Design, FMCAD 2021, New Haven, CT, USA, October 19-22, 2021*. IEEE, 1–10. https://doi.org/10.34727/2021/isbn.978-3-85448-046-4_27

Klas Leino, Zifan Wang, and Matt Fredrikson. 2021. Globally-Robust Neural Networks. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 6212–6222. http://proceedings.mlr.press/v139/leino21a.html

Xuankang Lin, He Zhu, Roopsha Samanta, and Suresh Jagannathan. 2020. Art: Abstraction Refinement-Guided Training for Provably Correct Neural Networks. In *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020*. IEEE, 148–157. https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_22

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019). arXiv:1907.11692 http://arxiv.org/abs/1907.11692

Soenksen LR, Kassis T, Conover ST, Marti-Fuster B, Birkenfeld JS, Tucker-Schwartz J, Naseem A, Stavert RR, Kim CC, Senna MM, Avilés-Izquierdo J, Collins JJ, Barzilay R, and Gray ML. 2021. Using deep learning for dermatologist-level detection of suspicious pigmented skin lesions from wide-field images. *Science Translational Medicine* 13, 581 (2021).

Lilia Maliar, Serguei Maliar, and Pablo Winant. 2021. Deep learning for solving dynamic economic models. *Journal of Monetary Economics* 122 (2021), 76–101. https://doi.org/10.1016/j.jmoneco.2021.07.004

Matthew Mirman, Timon Gehr, and Martin T. Vechev. 2018. Differentiable Abstract Interpretation for Provably Robust Neural Networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018 (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, 3575–3583. http://proceedings.mlr.press/v80/mirman18b.html

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. 2022a. Fast Model Editing at Scale. In *International Conference on Learning Representations*. https://openreview.net/forum?id=0DcZxeWfOPt

Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D. Manning, and Chelsea Finn. 2022b. Memory-Based Model Editing at Scale. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA (Proceedings of Machine Learning Research, Vol. 162)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (Eds.). PMLR, 15817–15831. https://proceedings.mlr.press/v162/mitchell22a.html

Norman Mu and Justin Gilmer. 2019. MNIST-C: A Robustness Benchmark for Computer Vision. *CoRR* abs/1906.02337 (2019). arXiv:1906.02337 http://arxiv.org/abs/1906.02337

Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. 2022. PRIMA: General and Precise Neural Network Certification via Scalable Convex Hull Approximations. *Proc. ACM Program. Lang.* 6, POPL, Article 43 (jan 2022), 33 pages. https://doi.org/10.1145/3498704

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. *CoRR* abs/2203.02155 (2022). https://doi.org/10.48550/arXiv.2203.02155 arXiv:2203.02155

Alessandro De Palma, Harkirat S. Behl, Rudy Bunel, Philip H. S. Torr, and M. Pawan Kumar. 2021. Scaling the Convex Barrier with Active Sets. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=uQfOy7LrlTR

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. (2019), 8024–8035. https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1409.1556

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

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T. Vechev. 2018. Fast and Effective Robustness Certification. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (Eds.). 10825–10836. https://proceedings.neurips.cc/paper/2018/hash/f2f446980d8e971ef3da97af089481c3-Abstract.html

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. 2019. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.* 3, POPL (2019), 41:1–41:30. https://doi.org/10.1145/3290354

Anton Sinitsin, Vsevolod Plokhotnyuk, Dmitry V. Pyrkin, Sergei Popov, and Artem Babenko. 2020. Editable Neural Networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020.* OpenReview.net. https://openreview.net/forum?id=HJedXaEtvS

Matthew Sotoudeh, Zhe Tao, and Aditya V. Thakur. 2023. SyReNN: A tool for analyzing deep neural networks. *Int. J. Softw. Tools Technol. Transf.* 25, 2 (2023), 145–165. https://doi.org/10.1007/s10009-023-00695-1

Matthew Sotoudeh and Aditya V. Thakur. 2019. Correcting Deep Neural Networks with Small, Generalizing Patches. In *NeurIPS 2019 Workshop on Safety and Robustness in Decision Making.*

Matthew Sotoudeh and Aditya V. Thakur. 2021a. PRDNN. https://github.com/95616ARG/PRDNN.

Matthew Sotoudeh and Aditya V. Thakur. 2021b. Provable repair of deep neural networks. In *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, Stephen N. Freund and Eran Yahav (Eds.). ACM, 588–603. https://doi.org/10.1145/3453483.3454064

Matthew Sotoudeh and Aditya V. Thakur. 2021c. SyReNN: A Tool for Analyzing Deep Neural Networks. In *27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS*. Springer. https://doi.org/10.1007/978-3-030-72013-1_15

Rick Stevens, Valerie Taylor, Jeff Nichols, Arthur Barney Maccabe, Katherine Yelick, and David Brown. 2020. *AI for Science.* Technical Report. Argonne National Lab.(ANL), Argonne, IL (United States).

Zhe Tao, Stephanie Nawas, Jacqueline Mitchell, and Aditya V. Thakur. 2023. Architecture-Preserving Provable Repair of Deep Neural Networks. *CoRR* abs/2304.03496 (2023). arXiv:2304.03496 http://arxiv.org/abs/2304.03496

Muhammad Usman, Divya Gopinath, Youcheng Sun, Yannic Noller, and Corina S. Pasareanu. 2021. NNrepair: 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 (Lecture Notes in Computer Science, Vol. 12759)*, Alexandra Silva and K. Rustan M. Leino (Eds.). Springer, 3–25. https://doi.org/10.1007/978-3-030-81685-8_1

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Efficient Formal Safety Analysis of Neural Networks. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (Eds.). 6369–6379. https://proceedings.neurips.cc/paper/2018/hash/2ecd2bd94734e5dd392d8678bc64cdab-Abstract.html

Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. 2021. Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021.* OpenReview.net. https://openreview.net/forum?id=nVZtXBI6LNn

Received 2022-11-10; accepted 2023-03-31