Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Repair Instead of Retraining: A Constraint-Guided Framework for Neural Network Repair

Ting Yu Liu
National Cheng Chi University
Fang Yu
yuf@nccu.edu.tw

National Cheng Chi University

Research Article

Keywords: neural network repair, explainable AI, DeepSHAP, concolic execution, Max-SMT, Z3

Posted Date: July 22nd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10274597/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Repair Instead of Retraining: A Constraint-Guided Framework for Neural Network Repair

Ting-Yu Liu[1*] and Fang Yu[1]

[1*]Department of Information Management, National Cheng Chi University, Taipei, Taiwan.

*Corresponding author(s). E-mail(s): 113356048@g.nccu.edu.tw;
Contributing authors: yuf@nccu.edu.tw;

Abstract

Deep neural networks remain vulnerable to adversarial perturbations and backdoor attacks, yet repairing a deployed model without full retraining must balance interpretability, behavioral fidelity, and computational cost. We present a constraint-guided—but not formally certified—framework that uses DeepSHAP to localize a sparse set of fault-relevant weights, collects symbolic path constraints from benign and adversarial executions through concolic testing, and searches for weight updates with Max-SMT. Candidate repairs are always evaluated on the original deployed model, so reported accuracy and attack success rate reflect actual runtime behavior rather than simplified analysis models. Rather than committing to a single fixed recipe, we characterize a configurable repair design space—including which weights to modify, how constraints are collected, and how candidates are ranked—and study it through more than 4,700 runs on six benchmarks. Repair success depends critically on jointly choosing the weight direction, the surrogate model used for constraint collection, and the simplification strength. For backdoor benchmarks, outgoing-weight repair with an activation-gated surrogate achieves near-complete backdoor removal on Fashion-MNIST and MNIST-BD (attack success rate from above 98% to below 8%). For adversarial benchmarks, bias-only repair substantially reduces misbehavior on MNIST-6 (100% to 5.81%) and ResNet18 (85.51% to 40.42%). CIFAR-10 and GTSRB remain challenging under single-layer repair, highlighting a tractability–fidelity trade-off. These results show that bias-direction repair is a critical and previously underexplored option for adversarial correction, discoverable only through systematic design-space exploration.

Keywords: neural network repair, explainable AI, DeepSHAP, concolic execution, Max-SMT, Z3

1 Introduction

Deep neural networks (DNNs) have achieved remarkable success across a wide range of applications, including image recognition, machine translation, medical diagnosis, and autonomous driving Gulshan et al. (2016); Angelova et al. (2015); Graves et al. (2013); Bahdanau et al. (2015). Despite these advances, modern DNNs remain vulnerable to failures such as misclassifications, adversarial perturbations, and backdoor triggers, which can lead to severe consequences in safety-critical settings Szegedy et al. (2013); Goodfellow et al. (2014); Gu et al. (2019). Improving the reliability of deployed neural networks therefore remains an important and challenging problem.

Among these failure modes, adversarial examples and backdoor attacks are particularly concerning because small, often imperceptible input changes can cause a model to produce incorrect predictions with high confidence Szegedy et al. (2013); Goodfellow et al. (2014); Gu et al. (2019). Existing defenses include adversarial training, data augmentation, input transformations, and various post-processing techniques Amodei et al. (2016); Akhtar and Mian (2018). However, these approaches are fundamentally training-time strategies and are difficult to apply selectively once a model is already deployed. Retraining or fine-tuning updates many or all parameters, is computationally expensive at scale Strubell et al. (2019), and does not provide a controllable recipe for eliminating a specific observed misbehavior while preserving the rest of the model's decision logic; parameter changes may introduce broad and difficult-to-control behavioral drift beyond the originally observed faults. More broadly, recent studies on generalization suggest that additional training does not always yield controllable behavioral guarantees Yang et al. (2026), motivating *post-hoc neural network repair* that targets already observable failures in deployed models Sun et al. (2022, 2025); Usman et al. (2021); Tao et al. (2023); Ma et al. (2025b). Recent work spans three paradigms: grey-box repair methods (e.g., causality- or search-based weight adjustment), white-box constraint-based approaches (e.g., QP- or SMT-based optimization), and provable or verification-guided methods that offer formal correctness guarantees. While grey-box methods are flexible, they often provide limited insight into the internal decision logic of the repaired model. Constraint-based and provable methods offer stronger logical control, but their scalability depends heavily on (i) how precisely the repair space is localized, and (ii) how tractable the resulting symbolic constraints are. Recent verification-guided approaches such as VeRe Ma et al. (2024) improve localization precision via formal analysis, and interpretability-based methods such as INNER Chen et al. (2024) leverage model probes to identify fault neurons. However, these methods either require architectural transformations, manually specified formal properties, or lack a mechanism to directly encode execution-level behavioral specifications for adversarial correction. In particular, directly encoding a full-scale neural network into a constraint system can quickly become intractable unless the repair is carefully localized and the symbolic formulas are simplified.

Rather than relying on additional training to wash out misbehavior, we formulate correction as a configurable, logic-aware repair problem over localized model parameters. Our framework is *constraint-guided rather than formally certified*: Max-SMT synthesizes candidate repairs under collected path constraints, and deployable behavior is assessed empirically through faithful validation on N, not through polytope-level

or architecture-preserving correctness proofs. We address these challenges through four coupled stages: *fault localization*, *repair-target selection*, *path-constraint specification* (including rigid simplification and ternary network formulations), and *constraint-guided repair with faithful validation*. Unlike provable repair systems that require architectural changes or manually specified polytope constraints, and unlike grey-box methods that lack interpretability, our approach combines the transparency of XAI with symbolic path reasoning. First, we use DeepSHAP Lin and Yu (2023) to derive neuron-level attribution masks under benign and adversarial inputs; we primarily target *attack-only* neurons (high attribution under adversarial inputs but not under benign inputs), while also exploring intersection-based masks as a design-space dimension. Second, we refine localization into a sparse repair mask over incoming weights, outgoing weights, and biases, with column-wise selection strategies (percentile thresholds or fixed random budgets) that control the number of symbolic repair variables. Third, we define repair intent as *path-level* constraints on the deployment network N: benign traces yield hard constraints that preserve correct internal branching, while adversarial traces yield mutually exclusive soft alternative paths. Because concolic execution on N alone often yields formulas too large for Max-SMT, we collect path constraints on analysis surrogates N_w^Δ and N_i^Δ: the former applies ternary gating to non-repair weights, the latter activation gating to non-repair edges, while repair weights remain exact symbolic unknowns in both cases. Fourth, we solve a Top-K Max-SMT problem over the repair mask, generate diverse candidates via blocking clauses, and rank them using a balanced score computed under *faithful validation* on N—ensuring that reported accuracy and attack success rate reflect deployable behavior rather than surrogate-analysis artifacts.

Overall, we treat repair as an empirically explored design space rather than a single fixed heuristic, and report the best configurations found under a unified evaluation protocol. Our study exposes an explicit tractability–fidelity trade-off: stronger simplification (larger Δ) reduces constraint size and solver time but can distort repair guidance when calibrated poorly.

The contributions of this paper are summarized as follows:

- We propose a configurable logic-aware repair framework combining DeepSHAP localization, path-level hard/soft concolic specifications, ternary and activation-gated analysis surrogates (N_w^Δ, N_i^Δ), and Top-K Max-SMT with faithful validation on N.
- We formalize a repair *design space* over neuron source (attack-only vs. intersection), weight direction (incoming, outgoing, bias), and column-wise selection (percentile vs. random budget), instantiated via precomputed repair masks in our artifact.
- We introduce encoding-time rigid simplifications—common subexpression elimination, negation normalization, and QF_LRA–guided expression pruning—to keep concolic path constraints tractable for Max-SMT without altering repair semantics over Θ_S.
- We introduce ternary network formulation N_w^Δ (ternary $\{-1, 0, +1\}$ gating on non-repair weights) and complementary activation-gated surrogate N_i^Δ, alongside deployment network N as the semantic reference; all three propagate symbolic

expressions over Θ_S from ℓ^* to the output while applying different layer maps to non-repair edges.
- We establish a faithful validation protocol in which all reported repair metrics are measured on N with exact affine operations, regardless of whether constraints were collected on N, N_w^Δ, or N_i^Δ.
- We empirically characterize tractability–fidelity trade-offs across MNIST and Fashion-MNIST benchmarks, including constraint-count-driven solver cost and best explored repairs (e.g., Fashion-MNIST ASR 99.77%→6.67%).

2 Related Work

Neural network repair has been studied from several complementary directions, including heuristic search-based repair, constraint-based repair, verification-guided repair, and explanation-guided fault localization. Our work is positioned at the intersection of these directions. It uses contrastive DeepSHAP attribution to localize repair-relevant neurons and weights, concolic execution to extract symbolic path constraints, and Max-SMT optimization to synthesize sparse weight modifications for adversarial repair.

2.1 Neural Network Repair

Neural network repair aims to modify an existing trained model so that undesirable behaviors are reduced while the original correct behavior is preserved. Existing repair approaches differ in how they localize faulty components, how they formulate the repair objective, and whether they provide formal guarantees.

Orthogonal to specific repair algorithms, Yang et al. analyze when training can guarantee behavioral generalization beyond finite training data, showing that Transformers in general admit no computable non-asymptotic length-generalization bound and that fixed-precision variants may require exponentially long training examples in the worst case Yang et al. (2026). Their results concern training-time guarantees rather than post-deployment correction; our work addresses the complementary setting in which a trained model already exhibits observable adversarial or backdoor misbehavior and must be repaired through localized, constrained weight updates.

A representative search-based repair method is CARE Sun et al. (2022), which performs causality-based fault localization to identify neurons that contribute most to undesirable behavior and then applies Particle Swarm Optimization (PSO) to adjust the parameters associated with those neurons. CARE has been evaluated on fairness improvement, backdoor removal, and safety-property repair tasks, demonstrating that targeted weight modification can repair multiple classes of neural network defects without full retraining. However, although CARE provides a causal rationale for neuron selection, the subsequent PSO-based repair remains heuristic and does not explicitly encode path-level symbolic constraints.

Building on causality-based localization, CCBR Liu et al. (2024) combines counterfactual causal inference with multi-objective optimization to improve faulty neuron localization and repair effectiveness. Vares and Johnson Vares and Johnson (2025) further discuss the potential and limitations of causality-driven neural network repair,

particularly when misbehavior arises from complex multi-layer interactions. These approaches are closely related to our motivation of avoiding full retraining and focusing repair on a small set of influential model components. In contrast, our approach uses DeepSHAP attribution differences between paired benign and adversarial inputs as the localization signal, and then formulates the repair step as a symbolic Max-SMT optimization problem rather than a purely heuristic search.

Another related direction is interpretability-based repair. INNER Chen et al. (2024) introduces neuron routing and model probes to identify neurons responsible for backdoor behavior, then adjusts the selected neurons to suppress undesired activations while maintaining class balance. Like INNER, our work uses interpretability to guide repair. However, INNER focuses on probe-based semantic explanations and direct parameter adjustment, whereas our method uses contrastive DeepSHAP attribution shifts to construct a sparse repair mask and enforces repair objectives through symbolic path constraints.

2.2 XAI-Guided Fault Localization and Adversarial Detection

Explainable AI has been widely used to analyze the internal behavior of neural networks. SHAP and DeepSHAP Lundberg and Lee (2017) provide attribution scores that quantify how input features or internal neurons contribute to a model prediction. Prior work has shown that such explanation patterns can reveal differences between benign and adversarial inputs. In particular, Lin and Yu Lin and Yu (2023) use DeepSHAP values to compute neuron contributions layer by layer and show that normal and adversarial examples induce different attribution distributions. They further summarize critical neurons into decision-logic representations for adversarial-example detection.

This line of work is important for our method because it suggests that adversarial behavior is reflected not only in the output label but also in the model's internal attribution structure. However, most XAI-based adversarial methods use explanations for detection rather than repair. That is, they determine whether an input is likely adversarial, but they do not modify the model to reduce adversarial vulnerability. Our work repurposes this observation for repair: instead of using DeepSHAP signatures as detector features, we use contrastive attribution differences between benign and adversarial pairs to identify repair-relevant neurons and weights. These localized components then become the variables of a symbolic repair problem.

This distinction is central to our contribution. Existing XAI-guided adversarial detection methods reveal that benign and adversarial inputs follow different internal decision logic, while our framework uses that difference as a fault-localization signal. In other words, DeepSHAP is not used as a post-hoc detector in our work; it is used to define a sparse and interpretable repair target.

2.3 Constraint-Based Neural Network Repair

Constraint-based repair methods leverage access to the model structure and parameters to synthesize weight modifications that satisfy desired behavioral constraints. NNRepair Usman et al. (2021) formulates repair as a constraint-solving problem

and uses activation patterns at intermediate layers to guide repair. It has been applied to adversarial robustness and data-poisoning scenarios, but its localization and repair are typically defined over selected layers and rely on user-provided correctness specifications.

Other white-box approaches formulate repair as mathematical optimization. Majd et al. Majd et al. (2021) encode local neural network repair as a Mixed Integer Quadratic Programming (MIQP) problem, aiming to minimize parameter changes while satisfying safety constraints. AutoRIC Sun et al. (2025) formulates repair as constrained quadratic programming and reduces solving cost through constraint selection. These methods provide precise optimization formulations, but they often require carefully designed constraints and can become expensive as the model and repair region grow.

Our work is related to these approaches in that it also performs white-box parameter repair. However, we differ in two aspects. First, instead of manually selecting the repair region or relying only on activation-based localization, we use DeepSHAP attribution shifts to construct an interpretable repair mask. Second, instead of encoding the entire repair problem as QP, MIQP, or LP, we collect path-level symbolic constraints through concolic execution and solve a sparse Max-SMT problem over the selected weights. This allows the repair objective to remain explicit while keeping the search space small.

2.4 Provable and Verification-Guided Repair

Provable repair methods aim to guarantee that a repaired network satisfies a formal specification over a given input region. PRDNN Sotoudeh and Thakur (2021) introduces a decoupled neural network representation and reduces provable repair to linear programming. APRNN Tao et al. (2023) extends this direction by preserving the original architecture while supporting efficient polytope repair. REASSURE Fu and Li (2022) further studies sound and complete repair under formal specifications.

More recent verification-guided approaches improve scalability and generality. VeRe Ma et al. (2024) uses linear relaxation to estimate the repair significance of neurons and optimizes problematic neurons without relying on heuristic layer selection. ProRepair Ma et al. (2025a) synthesizes a preimage proxy to support efficient point-wise and region-wise repair. PatchPro Chi et al. (2025) repairs properties by synthesizing patch modules that are integrated with the original network. DefEnSyn Yuviler and Drachsler-Cohen (2025) synthesizes short repair programs that adjust a small number of neurons to improve robustness.

These methods provide stronger formal guarantees than our framework, but they also rely on explicit formal input-output specifications, region definitions, architectural transformations, or additional patch/program synthesis. Our objective is different: we target localized adversarial repair over empirically observed benign–adversarial discrepancies rather than global certified repair over a fully specified input region. This design avoids the need for manually specified polytopes or architecture-level transformations while still retaining an explicit symbolic optimization objective through Max-SMT.

2.5 Program Repair and Symbolic Constraint Extraction

Automated program repair (APR) aims to automatically generate patches for buggy programs based on test cases, specifications, or semantic constraints. Existing APR techniques have been developed along several directions, including search-based repair, template-based repair, learning-based repair, and constraint-based repair. Among these directions, constraint-based and semantics-based repair are the most relevant to our work, because they typically transform observed program behaviors into symbolic repair constraints and then synthesize patches that satisfy those constraints.

Early generate-and-validate approaches, such as GenProg Le Goues et al. (2012), search over candidate patches and validate them against a given test suite. In contrast, semantics-based repair methods aim to derive constraints that characterize how a program should be modified. SemFix Nguyen et al. (2013), for example, combines symbolic execution, constraint solving, and program synthesis to infer repairs from failing and passing executions. DirectFix Mechtaev et al. (2015) further formulates repair as a constraint-solving problem and seeks patches that satisfy correctness constraints while minimizing syntactic changes. Angelix Mechtaev et al. (2016) improves scalability by extracting lightweight repair constraints from symbolic executions and using them to guide patch synthesis.

These program-repair techniques show that symbolic constraints can serve as an intermediate representation between observed failures and candidate repairs. Our work follows this high-level principle, but applies it to neural network repair rather than conventional source-code repair. Instead of synthesizing source-code patches, we use concolic execution to extract path constraints from neural network executions and then synthesize sparse modifications to selected neural weights. In this sense, the repair variables in our framework are not program expressions or statements, but weight parameters localized by contrastive DeepSHAP attribution.

PyCT Chen et al. (2021) is used in our framework as a concolic execution engine for collecting symbolic path constraints from Python-level neural network computations. This use of PyCT is inspired by the program-repair tradition of converting faulty executions into symbolic repair constraints. However, our goal is not to build a general-purpose Python program repair system. Rather, we adapt the constraint-extraction idea to adversarial neural network repair, where benign–adversarial execution discrepancies provide the repair evidence, DeepSHAP identifies the repair-relevant weights, and Max-SMT synthesizes candidate weight updates under hard and soft path constraints.

2.6 Position of Our Work

In summary, existing neural network repair methods either rely on heuristic optimization after fault localization, require explicit formal specifications for provable repair, or introduce architectural or patch-based modifications. Existing XAI-based adversarial methods reveal explanation-level differences between benign and adversarial inputs, but typically stop at detection. Existing concolic testing methods explore DNN behaviors and generate tests, but are not primarily designed for sparse weight repair.

Our framework bridges these directions by combining contrastive DeepSHAP-based localization, concolic path-constraint extraction, and sparse Max-SMT optimization within a single adversarial repair pipeline. Compared with grey-box repair methods such as CARE Sun et al. (2022) and CCBR Liu et al. (2024), our method gives an explicit attribution-based rationale for selecting repair targets and encodes the repair objective symbolically. Compared with provable repair systems such as PRDNN Sotoudeh and Thakur (2021), APRNN Tao et al. (2023), and VeRe Ma et al. (2024), our method does not require architecture transformation or manually specified polytope regions. Compared with interpretability-based repair such as INNER Chen et al. (2024), our method does not directly update parameters from explanation scores; instead, it uses explanations to localize repair variables and then synthesizes candidate repairs through Max-SMT.

The key distinction of our work is therefore not any single component alone, but the integration of three components: (1) contrastive DeepSHAP attribution for identifying adversarial-specific neurons and weights; (2) concolic execution for extracting symbolic path constraints from observed benign and adversarial behaviors; and (3) Top-K Max-SMT solving with faithful validation to generate and select sparse weight modifications. This combination provides an interpretable and empirically tractable repair framework for adversarial neural network repair.

3 Methodology

This section presents our configurable repair framework. Given a trained network N and paired benign/adversarial samples, we seek a repaired N^* that preserves benign behavior while reducing adversarial misclassification by modifying only a sparse repair mask Θ_S. A run is specified by a configuration

$$\mathcal{C} = (\text{neuronSrc},\ \ell^*,\ \alpha,\ \text{dir},\ \text{sel},\ \text{form},\ \Delta,\ K,\ |\mathcal{D}|),$$

where $\text{neuronSrc} \in \{\text{attack-only}, \text{intersection}\}$, $\text{dir} \in \{\text{incoming}, \text{outgoing}, \text{bias}\}$ selects *one* mask direction per run, sel chooses column-wise weights (percentile or random budget), $\text{form} \in \{N, N_w^\Delta, N_i^\Delta\}$ (equivalently normal, weight-δ, or input-δ in experiments), and $|\mathcal{D}|$ is the number of constraint-collection inputs per class.

Figure 1 summarizes the pipeline: localization and mask construction (Stages 1–2), path-constraint collection on N, N_w^Δ, or N_i^Δ (Stage 3), and Max-SMT repair with faithful validation on N (Stage 4). Surrogate networks are used for *analysis-time* constraint collection only, not deployment; deployable behavior is always judged on N with exact affine operations.

3.1 Fault Localization

Given the original model N, we partition the standard test split into a repair pool $\mathcal{P}$ and a disjoint evaluation set $\mathcal{T}$. Fault localization and all subsequent repair stages use only $\mathcal{P}$; final Acc/ASR in Section 4 are computed solely on $\mathcal{T}$. Let $\mathcal{D} = \mathcal{D}_{\text{norm}} \cup \mathcal{D}_{\text{adv}} \subseteq \mathcal{P}$, where $\mathcal{D}_{\text{norm}}$ consists of benign samples and $\mathcal{D}_{\text{adv}}$ is generated from the same samples by applying the target attack. The fault localization stage identifies *where* in

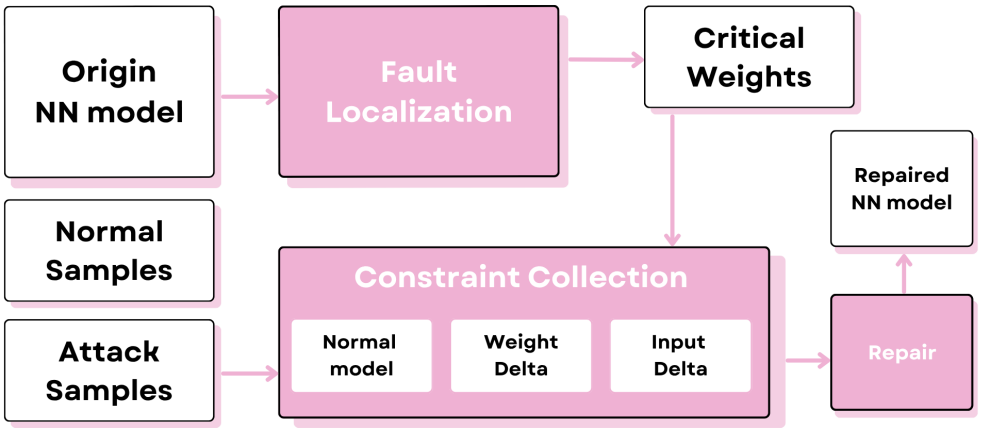


Fig. 1 Overview of the configurable repair framework. DeepSHAP produces neuron masks and a repair weight mask Θ_S. Path-level hard/soft specifications are collected by concolic execution on the deployment network N or on analysis surrogates N_w^{Δ} and N_i^{Δ}. Top-K Max-SMT yields candidate repairs that are ranked by *faithful validation* on N.

the network the repair should be applied. It proceeds in two steps: first localizing the critical neurons responsible for adversarial behavior via DeepSHAP attribution, then deriving a sparse weight mask that defines the actual repair target.

3.1.1 Critical Neuron Identification via DeepSHAP

To minimize unnecessary structural perturbation and preserve the original decision-making logic of the model, we first identify a sparse subset of neurons at the target layer that are most relevant to adversarial vulnerability. Our design is inspired by the DeepSHAP-based adversarial analysis framework of Lin and Yu Lin and Yu (2023), which shows that normal and adversarial examples exhibit distinguishable attribution patterns across internal layers. Rather than using such attribution differences for adversarial detection, we repurpose them as fault-localization signals for repair.

Let $\phi_j(x) \in \mathbb{R}$ denote the DeepSHAP attribution of neuron j at the target layer for input x. For each input x, we first compute a per-image binary indicator by selecting the top-$\beta\%$ neurons ranked by absolute attribution:

$$m_j(x) = \mathbf{1}\big[|\phi_j(x)| \geq \tau_\beta(x)\big],$$

where $\tau_\beta(x)$ is the $(1-\beta)$-quantile of $\{|\phi_j(x)|\}$ over all neurons j in the target layer (default β=5). We then aggregate these per-image indicators over each input set:

$$\bar{m}_j^{\text{norm}} = \frac{1}{|\mathcal{D}_{\text{norm}}|} \sum_{x \in \mathcal{D}_{\text{norm}}} m_j(x), \qquad \bar{m}_j^{\text{adv}} = \frac{1}{|\mathcal{D}_{\text{adv}}|} \sum_{x \in \mathcal{D}_{\text{adv}}} m_j(x).$$

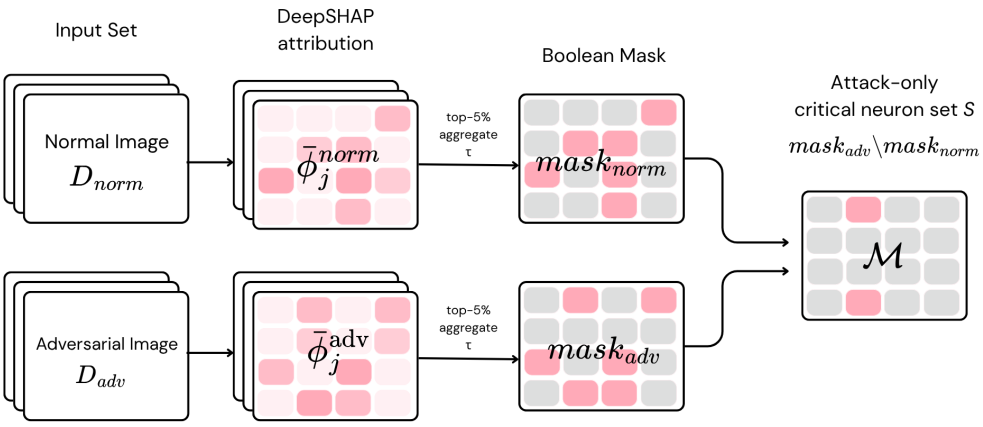


Fig. 2 DeepSHAP-based critical neuron identification. Mean absolute attributions $\bar{\phi}_j^{\text{norm}}$ and $\bar{\phi}_j^{\text{adv}}$ are each thresholded at their respective $(1-\alpha)$-quantile $(\tau_{\text{norm}}, \tau_{\text{adv}})$ to produce boolean masks. The attack-only set $S = \text{mask}_{\text{adv}} \setminus \text{mask}_{\text{norm}}$ (pink cells) captures neurons activated specifically under adversarial inputs and is passed as the repair mask $\mathcal{M}$ to the constraint engine.

A neuron j is included in the final boolean mask if its aggregated score exceeds the shared threshold τ:

$$\text{mask}_{\text{norm}} = \{j : \bar{m}_j^{\text{norm}} \geq \tau\}, \qquad \text{mask}_{\text{adv}} = \{j : \bar{m}_j^{\text{adv}} \geq \tau\}.$$

Note that the same threshold τ is applied to both input sets, ensuring that the two masks are comparable.

The critical neuron set for repair is defined as the *attack-only* neurons—those flagged as important under adversarial inputs but *not* under benign inputs (Figure 2):

$$S = \text{mask}_{\text{adv}} \setminus \text{mask}_{\text{norm}}.$$

This criterion deliberately excludes neurons that are influential under both conditions, targeting instead the subset whose high attribution is *specific to adversarial failure*. Neurons important in both regimes are presumed to encode correct decision logic and are left untouched to preserve clean accuracy.

3.1.2 Repair Mask Generation

Given the critical neuron set S identified in Section 3.1.1, we derive a sparse binary mask $\mathcal{M}$ over network parameters that defines the repair target passed to the constraint engine. The mask is characterized by two orthogonal choices: a *weight direction* dir and a *column-wise selection policy* sel.

Weight direction.

We distinguish three directions relative to the target layer (illustrated in Figure 3):

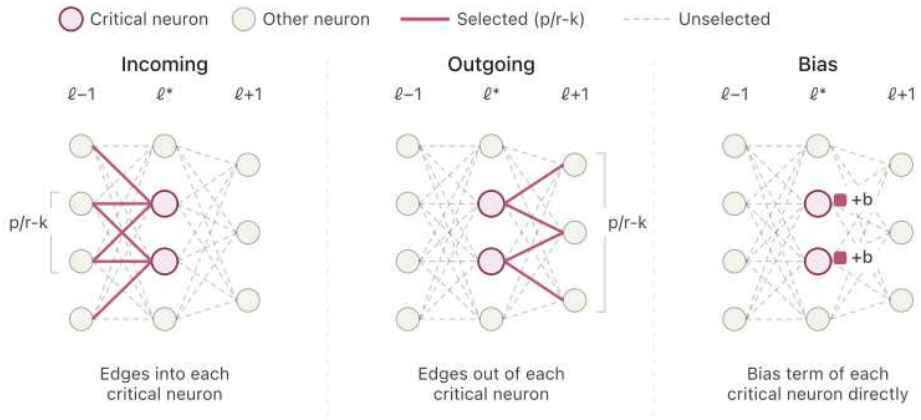


Fig. 3 Three repair-mask directions on a sample network with two critical neurons (pink). **Incoming**: a p/r-k-selected subset of edges arriving at each critical neuron is included in $\mathcal{M}$. **Outgoing**: a p/r-k-selected subset of edges leaving each critical neuron is included. **Bias**: only the bias term b of each critical neuron is modified.

- Θ_S^{in}: weights *incoming* to each critical neuron from the preceding layer. Modifying these controls what pre-activation value the neuron receives, directly targeting the fault-relevant signal path.
- Θ_S^{out}: weights *outgoing* from each critical neuron to the subsequent layer. Modifying these controls how the neuron's activation propagates downstream.
- Θ_S^{b}: the *bias* of each critical neuron. No weight edges are included; only the scalar offset is adjusted.

Each repair run selects *exactly one* direction: dir is incoming, outgoing, or bias, but never a combination thereof at the same layer.

Selection policy.

Because including all weights in a direction would produce prohibitively large SMT formulas, a column-wise policy sel further prunes each critical-neuron column. *Percentile* selection (pq) retains only weights whose magnitude exceeds the q-th percentile within that column (default q=95 in our experiments). *Random-k* selection (r-k) draws k weights per column uniformly at random, enforcing a fixed symbolic budget (e.g., k=50 in our cross-model study). The resulting mask $\mathcal{M}$ is passed to the concolic engine.

3.2 Path-Constraint Specification

Repair intent is expressed as path-level constraints on the original network N. We treat weights in Θ_S as *concolic variables* (concrete value plus symbolic identifier) and perform forward propagation from ℓ^* to the output using PyCT Chen et al. (2021). For each input x, every ReLU in the propagation path records a branch predicate over the repair variables, forming a path trace π_x.

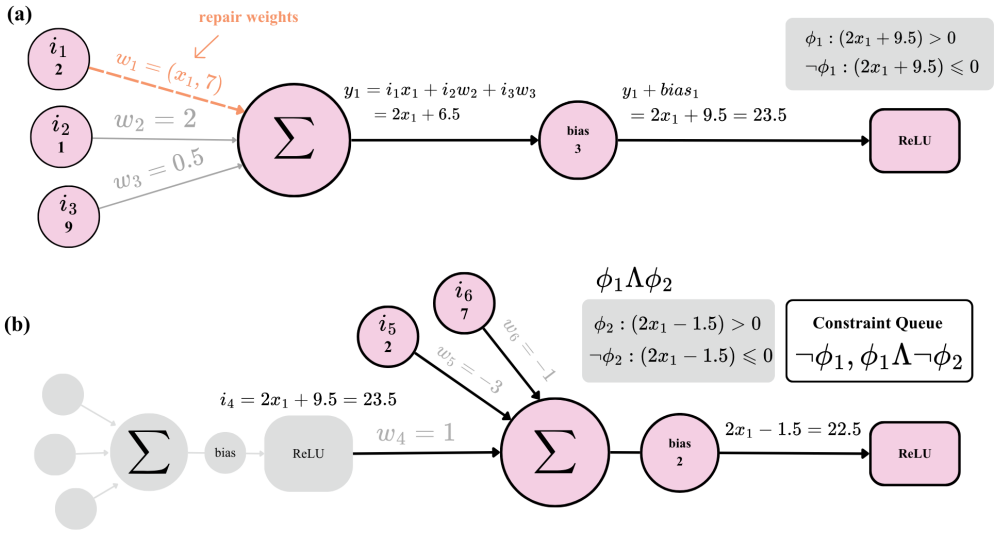


Fig. 4 Running example of concolic path extraction on the deployment network N. (a) Repair weight $w_1{=}(x_1, 7)$ is concolic (orange); forward propagation from ℓ^* yields pre-activation $2x_1{+}9.5$ and branch predicate $\phi_1{:}(2x_1{+}9.5){>}0$. (b) A second ReLU adds ϕ_2; PyCT's negation-flip enqueues $\neg\phi_1$ and $\phi_1{\wedge}\neg\phi_2$ as alternative paths.

Scope of symbolic variables.

Our SMT instance introduces unknowns only for the repair mask $\Theta_S \subseteq \Theta_{\ell^*}$ at a single target layer ℓ^*. All other weights and all activations before ℓ^* are fixed to their concrete values for each collected input. From ℓ^* onward, concolic execution propagates *symbolic expressions* over Θ_S through subsequent layers: downstream activations are not additional decision variables, but functions of the form $\mathrm{ReLU}(\mathrm{affine}(\Theta_S))$ multiplied by concrete downstream weights. In our reported experiments, this *single-layer, single-direction* repair policy—one of incoming, outgoing, or bias at ℓ^* per run, never mixed at the same layer—keeps branch predicates along a fixed ReLU path linear in Θ_S, so we instantiate Max-SMT in QF_LRA Sun et al. (2018) (Section 4.3). Bilinear constraints arise when repair unknowns span multiple layers—regardless of whether each layer uses incoming, bias, or outgoing weights—because deeper layers then multiply repair unknowns by activations that already depend on upstream repair unknowns (e.g., $w^{(\ell+1)} \cdot \mathrm{ReLU}(w^{(\ell)}x)$); the same effect occurs if multiple directions at ℓ^* were combined in one mask, which our configuration excludes. We treat such multi-layer or mixed-direction symbolic repair as future work (Section 5).

Under the deployment network N, pre-activations are computed as exact products:

$$z = \sum_{i \in \Theta_S} w_i \cdot x_i + \sum_{j \notin \Theta_S} \bar{w}_j \cdot x_j,$$

where repair weights w_i are symbolic unknowns, non-repair weights $\bar{w}_j$ are concrete constants fixed at execution time, and all inputs x_i to the layer are concrete.

Hard and soft constraints.

For each $x \in \mathcal{D}_{\text{norm}}$, we conjoin all predicates along the observed benign trace into a path trace and then into a hard constraint:

$$\pi_x = \bigwedge_{i=0}^{B-1} \phi_i, \quad B = |\pi_x|, \qquad C_{\text{hard}} = \bigwedge_{x \in \mathcal{D}_{\text{norm}}} \pi_x.$$

For each $x \in \mathcal{D}_{\text{adv}}$, PyCT's negation-flip mechanism enumerates mutually exclusive alternative paths: starting from the current trace $\phi_0 \wedge \cdots \wedge \phi_{B-1}$, each alternative is obtained by retaining a prefix of the original predicates and negating the next one. The disjunction over all such alternatives,

$$\alpha_x = \bigvee_{i=0}^{B-1} \Big(\bigwedge_{j=0}^{i-1} \phi_j \Big) \wedge \neg \phi_i,$$

expresses the wish that the adversarial input is rerouted away from at least one branch on its current execution path. The soft constraint is then

$$C_{\text{soft}} = \bigwedge_{x \in \mathcal{D}_{\text{adv}}} \alpha_x.$$

Each α_x is submitted as an independent soft clause via Z3's `assert-soft` mechanism, allowing the Max-SMT solver to maximize the number of satisfied alternative-path conditions. However, concolic execution on N retains every non-repair term $\bar{w}_j \cdot x_j$ explicitly in each predicate, which for wide or deep target layers yields formula sizes that are costly or intractable for Max-SMT. Section 3.3 applies encoding-time rigid simplifications to shrink the SMT instance without changing repair semantics; Section 3.4 further defines analysis surrogates N_w^{Δ} and N_i^{Δ} when formulas remain too large.

3.3 Rigid Simplification

Path predicates collected by concolic execution are first lowered to a Max-SMT instance in QF_LRA before solving. Even on the deployment network N, naively expanding every branch condition can produce formulas whose size and parsing cost dominate wall-clock time, because the same layer-wise affine computations are shared across many predicates in C_{hard} and C_{soft}. We therefore apply three encoding simplifications after predicate generation and before Max-SMT solving. Following the preprocessing taxonomy of Bjørner and Fazekas Bjørner and Fazekas (2023), we separate *semantics-preserving* transforms (common subexpression elimination and negation normalization) from expression pruning, which restricts symbolic dependencies to QF_LRA expressible repair boundaries and is analogous in role—though not identical in mechanism—to theory-lowering strengthenings that trade symbolic completeness for tractability.

Common subexpression elimination.

Concolic path constraints over neural networks exhibit heavy structural redundancy: the same affine pre-activations and shared intermediate expressions reappear across multiple branch predicates along a path and across samples in $\mathcal{D}$. Directly inlining these subexpressions inflates both formula length and Z3 parsing cost. After path predicates are expanded into SMT assertions but before the instance is passed to the solver, we perform structure-aware common subexpression elimination (CSE) on the resulting arithmetic terms Nightingale et al. (2015). Subexpressions that recur more than a length or frequency threshold are extracted into fresh auxiliary variables L_{tmp_i}, each defined by an equality constraint `(assert (=` L_{tmp_i} *expr*`))` and referenced in place of the shared subtree. This sharing does not alter the satisfiability of the repair problem over Θ_S: it is an equivalence-preserving definition introduction that reduces redundant syntax while keeping the encoded branch semantics unchanged.

Negation normalization.

Alternative-path constraints produced by PyCT's negation-flip mechanism often contain negated order comparisons, e.g. $\neg(e < 0)$. Such encodings are syntactically irregular for linear-arithmetic solvers. We therefore apply a lightweight negation normalization pass, rewriting eligible predicates into solver-friendly positive comparisons (e.g. $\neg(e < 0) \Rightarrow (e \geq 0)$). This step is semantics-preserving and inexpensive; in our experiments it has marginal impact on solver time but improves formula regularity and readability.

Expression pruning and repair boundaries.

Our concolic engine performs partial symbolic execution within QF_LRA. When an operation cannot be encoded in this theory, or when a neural activation approximation yields a concrete value on the current path (e.g., saturated sigmoid or tanh regions), the symbolic expression chain is pruned while the concrete execution result is preserved. Consequently, repair-relevant constraints are captured primarily at ReLU branch points and at pre-softmax logits rather than at post-softmax outputs: transcendental or otherwise non-linear activations can sever symbolic dependence on Θ_S, so post-softmax values may no longer carry repair variables even though the underlying classification intent remains expressible on logits. We do not implement the incremental preprocessing calculus of Bjørner and Fazekas Bjørner and Fazekas (2023); rather, we use their rigid versus under-/over-constrained taxonomy to clarify that expression pruning is a path-local abstraction at the concolic layer, distinct from the equivalence-preserving CSE and negation passes above. When rigid simplification alone does not bring the instance within budget, we additionally employ ternary and activation-gated surrogates (Section 3.4).

3.4 Ternary Network Formulations

When concolic execution on N still yields formulas too large for Max-SMT even after rigid simplification (Section 3.3), we collect path constraints on *analysis surrogates* that simplify only non-repair edges while keeping repair variables in Θ_S as exact

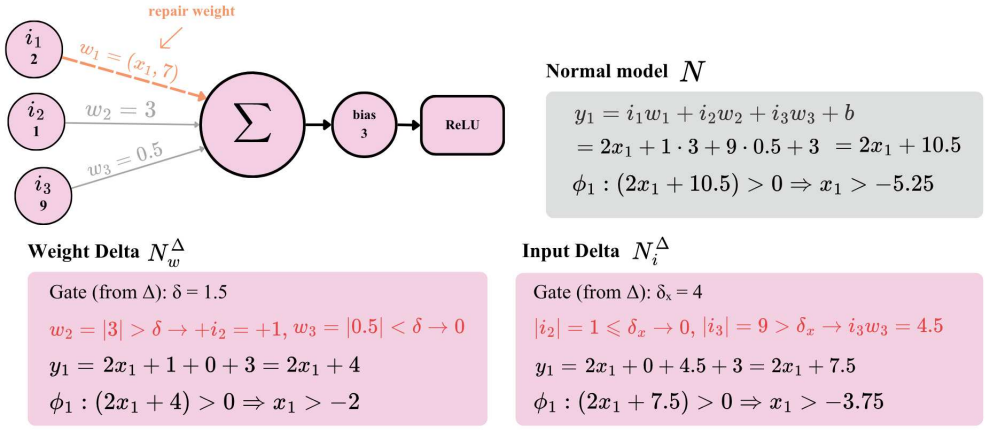


Fig. 5 Same neuron topology under three networks: deployment model N, ternary surrogate N_w^Δ (non-repair weights mapped to $\{-1, 0, +1\}$), and activation-gated surrogate N_i^Δ. Repair weight $w_1 \equiv x_1$ remains exact in all three; non-repair gating shifts the recorded path predicate ϕ_1. Constraints are *induced by* the chosen network, not by shortening formulas on N.

symbolic unknowns. We use two surrogates: N_w^Δ, obtained by a *ternary network formulation* that maps each non-repair weight to a coefficient in $\{-1, 0, +1\}$, and N_i^Δ, a complementary activation-gated surrogate that is not ternary. Path constraints are collected by concolic execution on the chosen network (N, N_w^Δ, or N_i^Δ). In all cases, each ReLU records a predicate of the form

$$\phi_k : \Big(\sum_{i \in \Theta_S} w_i x_i + C \Big) > 0,$$

where C aggregates all non-repair edge contributions and the bias under that network's layer map. On N, C is exact; on a surrogate, gating reduces C by pruning minor non-repair edges, shifting the ReLU boundary (Figure 5) while keeping the repair subspace exact.

For each collected normal sample $x \in \mathcal{D}_{\text{norm}}$, the hard constraint π_x freezes the ReLU branch decisions observed when x is executed on the chosen network under the current weights; by construction, x satisfies its collected predicates at collection time. We do *not* claim a formal equivalence between benign behavior on N and on N_w^Δ or N_i^Δ: surrogate formulations are tractability heuristics whose fidelity to deployable behavior is assessed empirically through faithful validation on N.

3.4.1 Ternary Surrogate N_w^Δ

Adapted from delta-based ternary quantization Li et al. (2022); Mellempudi et al. (2017-05-30), the ternary formulation defines N_w^Δ by gating each *non-repair* coefficient to a ternary value in $\{-1, 0, +1\}$ according to its magnitude. For each layer with n

non-repair weights, we compute a per-layer threshold

$$\delta = \frac{\Delta}{n} \sum_{i=1}^{n} |w_i|,$$

where Δ is a user-controlled scaling factor (the triangular superscript in N_w^{Δ}). The contribution of each non-repair weight is then determined by comparing w_j against δ:

$$\text{contrib}(x_j) = \begin{cases} +x_j, & w_j > \delta, \\ 0, & |w_j| \le \delta, \\ -x_j, & w_j < -\delta, \end{cases}$$

replacing the exact product $\bar{w}_j \cdot x_j$ with the ternary contribution $+x_j$, 0, or $-x_j$ (i.e., $\hat{w}_j \in \{+1, 0, -1\}$ times x_j). Repair weights in Θ_S are never gated; their contribution $w_i \cdot x_i$ remains exact. The pre-activation therefore takes the form

$$z = \sum_{i \in \Theta_S} w_i x_i + \sum_{\substack{j \notin \Theta_S \\ w_j > \delta}} x_j - \sum_{\substack{j \notin \Theta_S \\ w_j < -\delta}} x_j + b,$$

where the constant term $C = \sum_{j:w_j>\delta} x_j - \sum_{j:w_j<-\delta} x_j + b$ is the non-repair aggregate under N_w^{Δ}. A larger Δ raises δ, causing more non-repair weights to be gated to zero and further shortening each predicate, at the cost of greater deviation from the ReLU boundary of N. Concolic execution on N_w^{Δ} propagates symbolically from ℓ^* through all subsequent layers, so branch predicates capture the full depth of the surrogate's decision logic.

3.4.2 Activation-Gated Surrogate N_i^{Δ}

The activation-gated formulation defines N_i^{Δ}, a non-ternary surrogate in which non-repair edges are gated by activation magnitude. For each layer with n_{in} inputs, we compute

$$\delta_x = \frac{\Delta}{n_{\text{in}}} \sum_{j} |x_j|,$$

where Δ is the same user-controlled scaling factor as above. Each non-repair edge contribution is then gated by input magnitude:

$$\text{contrib}(x_j) = \begin{cases} 0, & |x_j| \le \delta_x, \\ w_j \cdot x_j, & \text{otherwise}, \end{cases}$$

retaining the exact product $w_j \cdot x_j$ for inputs that exceed the threshold, and discarding edges whose activations are small on this sample. Repair weights still use exact $w_i \cdot x_i$.

The pre-activation becomes

$$z = \sum_{i \in \Theta_S} w_i x_i + \sum_{\substack{j \notin \Theta_S \\ |x_j| > \delta_x}} w_j x_j + b.$$

A larger Δ raises δ_x, gating out more non-repair edges and further reducing the constant term C, again at the cost of greater boundary deviation from N. Unlike N_w^Δ, which drops edges based on weight magnitude regardless of the current input, N_i^Δ drops edges based on the actual activation of each sample, so the two surrogates prune different edges and yield different ReLU boundaries even on the same neuron (Figure 5). As in N_w^Δ, concolic execution on N_i^Δ propagates symbolically from ℓ^* through all subsequent layers; the two surrogates differ only in how non-repair edges are gated, not in symbolic unrolling depth.

The idea of combining localized symbolic repair with sample-dependent edge pruning is consistent with concolic testing for DNNs Sun et al. (2019). The resulting tractability–fidelity trade-off mirrors the abstraction cost studied in counterexample-guided DNN verification Liu et al. (2024): coarser simplification (larger Δ) reduces solver load but may introduce structural distortions that degrade repair quality, as we observe empirically in Section 4.

3.5 Evaluation

Given C_{hard} and C_{soft} from configuration $\mathcal{C}$, we solve a Max-SMT problem over repair variables in Θ_S only:

$$\begin{aligned} \text{find } \theta' \text{ such that } \; & C_{\text{hard}}(\theta') \text{ holds,} \\ & \text{while maximizing } \text{sat}\big(C_{\text{soft}}(\theta')\big). \end{aligned}$$

Constraints may be collected by concolic execution on N_w^Δ or N_i^Δ, but θ' always denotes real-valued weights in the deployment model N.

Top-K candidates

Satisfying surrogate-collected constraints is necessary but not sufficient for correct behavior on N. We therefore generate up to K diverse solutions using blocking clauses, instantiate each candidate on N, and rank the candidates using only the same small constraint-collection set $\mathcal{D} = \mathcal{D}_{\text{norm}} \cup \mathcal{D}_{\text{adv}}$ used to build the repair constraints. Specifically, we compute

$$Score = 0.5 \cdot Acc_{norm}^{\mathcal{D}}(N_i) + 0.5 \cdot Acc_{adv}^{\mathcal{D}}(N_i),$$

where both accuracies are evaluated on N with exact affine operations (i.e., on the deployment network N, not on a surrogate). This repair-time candidate selection step uses only the constraint-collection set $\mathcal{D} \subseteq \mathcal{P}$ and never touches the evaluation set $\mathcal{T}$.

After N^* is selected, we perform a separate post-repair evaluation on $\mathcal{T}$, disjoint from $\mathcal{P}$. The Acc and ASR reported in Section 4 and in the experiment tables are

computed solely on $\mathcal{T}$ and are never used for constraint generation or Top-K candidate selection within a run.

4 Experiments

We evaluate six benchmarks spanning two attack types, with 4,737 complete repair runs across a design space of analysis network (N, N_w^Δ, or N_i^Δ), direction, selection policy, abstraction strength Δ, target layer, and constraint-collection size. The following research questions structure our analysis:

RQ1: Overall effectiveness. Can the framework reduce ASR while preserving clean accuracy, evaluated on the full dataset?

RQ2: Repair-mask sensitivity. How do weight direction and column-wise selection policy affect repair quality, mask size, and wall-clock time?

RQ3: Surrogate-network trade-off. How do N, N_w^Δ, and N_i^Δ differ in accuracy preservation, constraint size, and wall-clock time?

RQ4: Abstraction strength. How does Δ under N_w^Δ affect the tractability–fidelity trade-off?

4.1 Experimental Setup

We evaluate four *backdoor* benchmarks (Fashion-MNIST, MNIST-BD, CIFAR-10, GTSRB) and two *adversarial* benchmarks (MNIST-6, ResNet18-CIFAR10). Table 1 lists model architectures and tasks; we abbreviate Fashion-MNIST as *Fashion* in all tables.

Each repair run fixes a single target layer ℓ^* and a single weight direction (incoming, outgoing, or bias). The design space $\mathcal{C}$ comprises: (i) analysis network (N, N_w^Δ, or N_i^Δ; reported as normal, weight-δ, or input-δ in tables); (ii) weight direction and selection policy (p95 or random-k, $k \in \{10, 25, 50\}$); (iii) abstraction strength $\Delta \in \{0, 1.0, 1.5, 2.0, 2.5, 3.0\}$; (iv) target layer ℓ^*; and (v) constraint-collection size $|\mathcal{D}|$.

Metrics.

We report post-repair clean accuracy (Acc) and attack success rate (ASR = 100% − repair_rate) on the evaluation set $\mathcal{T}$, disjoint from the repair pool $\mathcal{P}$ and constraint-collection set $\mathcal{D}$. For backdoor benchmarks, $|\mathcal{T}| = 1{,}000$; for adversarial benchmarks, $|\mathcal{T}|$ varies by dataset. Wall-clock time covers the complete pipeline (localization, constraint collection, Max-SMT solving, and candidate selection).

Reporting protocol.

Each configuration $\mathcal{C}$ defines one repair run. Within a run, Top-K candidate ranking uses only $\mathcal{D}$; $\mathcal{T}$ is not used for constraint generation or intra-run candidate selection. Table 3 reports the retrospectively best explored configuration $\mathcal{C}^*$ per benchmark—the completed run with lowest ASR on $\mathcal{T}$ subject to an accuracy drop of at most 5 percentage points among all runs in the design-space sweep. Because $\mathcal{C}^*$ is selected after exploring many configurations using the same $\mathcal{T}$, these headline results characterize repair *potential* under design-space exploration rather than a fully blind single-shot

Table 1 Benchmarks and model architectures.

Model	Architecture	Task
Fashion	Conv backbone + Dense(128) → Dense(10)	Backdoor
MNIST-BD	Conv backbone + Dense(512) → Dense(10)	Backdoor
CIFAR-10	Conv backbone + Dense(512) → Dense(10)	Backdoor
GTSRB	Conv backbone + Dense(512) → Dense(43)	Backdoor
MNIST-6	Conv backbone + Dense(50) → Dense(10)	Adversarial
ResNet18	ResNet-18 backbone + Dense(10)	Adversarial

deployment protocol. All other tables (RQ2–RQ4) report fixed configurations and should be read as controlled ablations within the same sweep.

Hardware.

Experiments run on Ubuntu 24.04.4 LTS with an Intel Core i7-13700 (24 threads), 16 GB RAM, and an NVIDIA Geforce RTX 4070Ti SUPER GPU.

4.1.1 Attack Construction and Repair-Time Data

Backdoor benchmark provenance.

The four backdoor benchmarks (Fashion-MNIST, MNIST-BD, CIFAR-10, GTSRB) use backdoored models inherited from the CARE benchmark Sun et al. (2022). Each model was injected following the BadNets methodology Gu et al. (2019): a 4×4 white-square trigger is placed at the bottom-right corner of each image, and a target label is chosen randomly per network. CARE varies the poisoning ratio to achieve a high pre-repair ASR while maintaining clean accuracy, rather than fixing a single poisoning rate across benchmarks; we therefore do not report a per-benchmark poisoning rate, but instead report the empirically measured pre-repair clean accuracy and ASR in our own evaluation environment as the Before values in all tables.

Reverse-engineered trigger protocol.

Consistent with CARE's evaluation setup, the backdoor trigger is treated as *unknown* at repair time. Both the constraint-collection adversarial inputs $\mathcal{D}_{\text{adv}}$ and the evaluation adversarial inputs $\mathcal{T}_{\text{adv}}$ are formed by applying a *reverse-engineered* trigger rather than the original training-time trigger. The reverse-engineered trigger is obtained via Neural Cleanse-style inversion on the inherited backdoored model. ASR is computed as the fraction of triggered test images classified into the attack target label under this reverse-engineered trigger.

Constraint-collection set construction.

We partition the standard test split into a repair pool $\mathcal{P}$ and a disjoint evaluation set $\mathcal{T}$. The constraint-collection set $\mathcal{D} \subseteq \mathcal{P}$ consists of two subsets:

- $\mathcal{D}_{\text{norm}}$: $|\mathcal{D}|$ clean images per class, sampled uniformly from $\mathcal{P}$.
- $\mathcal{D}_{\text{adv}}$: $|\mathcal{D}|$ triggered images per class, formed by applying the reverse-engineered trigger to the corresponding samples in $\mathcal{P}$.

Table 2 Sensitivity of repair quality to $|\mathcal{D}|$ (samples per class in constraint collection). Fixed config: MNIST-BD uses out/p95, i-δ, Δ=1.0; Fashion uses out/p95, i-δ, Δ=2.0.

$\lvert\mathcal{D}\rvert$	MNIST-BD		Fashion-MNIST	
	Acc (%)	ASR (%)	Acc (%)	ASR (%)
1	90.60	16.61	93.34	6.66
2	87.09	39.30	92.40	7.66
3	**99.29**	**7.05**	93.34	6.66
5	96.30	14.61	93.34	6.66
7	96.70	13.04	92.40	7.66
10	99.50	14.84	89.04	6.69

DeepSHAP localization likewise uses samples from $\mathcal{P}$ only. The same $\mathcal{P}/\mathcal{T}$ partition applies to adversarial benchmarks (MNIST-6, ResNet18), where $\mathcal{D}_{\text{adv}}$ consists of PGD adversarial examples generated from samples in $\mathcal{P}$. The parameter $|\mathcal{D}|$ is part of the design space $\mathcal{C}$ and is swept from 1 to 10 in all experiments. No sample in $\mathcal{T}$ is used for localization, constraint collection, or Top-K candidate ranking.

Sensitivity to $|\mathcal{D}|$.

Table 2 reports repair quality for varying $|\mathcal{D}|$ under the best-direction configuration on MNIST-BD and Fashion-MNIST, with all other hyperparameters fixed. On Fashion-MNIST, repair quality is highly stable: ASR remains at 6.66–6.70% across all $|\mathcal{D}|$ from 1 to 10, indicating that any small representative sample reliably covers the relevant fault execution paths. On MNIST-BD, the dependence is non-monotone: $|\mathcal{D}|$=2 produces *worse* ASR than $|\mathcal{D}|$=1, while $|\mathcal{D}|$=3 achieves the global best (7.05%); increasing $|\mathcal{D}|$ beyond 3 does not further improve results and modestly raises wall time. This non-monotone behavior reflects the fact that path coverage depends on *which* samples are drawn, not only how many; additional samples may share active paths with existing ones and add only redundant constraints. In practice, we recommend sweeping $|\mathcal{D}| \in \{2, \ldots, 5\}$ and selecting the candidate with the lowest post-repair ASR on a held-out validation split.

4.2 Overall Repair Performance

RQ1. Table 3 reports retrospectively best explored $\mathcal{C}^*$ for each benchmark (see Reporting protocol above). Backdoor and adversarial benchmarks are separated by a rule because they exhibit fundamentally different repair behavior.

Backdoor benchmarks.

Fashion-MNIST and MNIST-BD achieve near-complete backdoor removal. On Fashion-MNIST, ASR drops from 99.77% to **6.66%** while accuracy changes from 91.36% to 93.34% (160.4 s) under outgoing/p95 with $|\mathcal{D}|$=3. On MNIST-BD, ASR falls from 98.68% to **7.05%** with accuracy preserved at 99.29% (74.4 s). Critically, both best configurations use the `outgoing` direction on `dense_2` with surrogate N_i^{Δ} (input-δ)—not the incoming direction that is often assumed as default. CIFAR-10 yields partial success: ASR decreases from 88.34% to 68.75% at a cost of 2.05 pp accuracy

Table 3 RQ1 repair results: retrospectively best explored configuration per benchmark (Acc drop $\leq$5 pp; selected from design-space sweep using $\mathcal{T}$). ASR = 100%−Repair Rate; Time = total wall-clock time (s).

Model	Acc (%)		ASR (%)		Form.	Δ	Dir/Sel	$\lvert\mathcal{D}\rvert$	ℓ^*	Time (s)
	Before	After	Before	After						
Backdoor benchmarks										
Fashion	91.36	93.34	99.77	**6.66**	i-δ	2.0	out/p95	3	dense_2	160.4
MNIST-BD	99.09	99.29	98.68	**7.05**	i-δ	1.0	out/p95	3	dense_2	74.4
CIFAR-10	97.70	95.65	88.34	68.75	w-δ	1.5	out/p95	4	dense_2	408.2
GTSRB	96.51	92.89	90.48	85.96	normal	0.0	in/r-25	4	dense_1	694.2
Adversarial benchmarks										
MNIST-6	100.00	**100.00**	100.00	**5.81**	i-δ	2.0	bias	1	dense	0.51
ResNet18	89.49	89.02	85.51	**40.42**	i-δ	1.0	bias	3	dense	2.6

drop (408.2 s). GTSRB shows minimal improvement (ASR 90.48%→85.96%, 694.2 s), indicating a scalability boundary for single-layer repair.

Adversarial benchmarks.

The adversarial benchmarks reveal that *bias-direction repair* is the key design-space dimension for adversarial fault correction. On MNIST-6, selecting N_i^{Δ} with the `bias` direction (Δ=2.0, $|\mathcal{D}|$=1) achieves near-complete adversarial removal: ASR drops from 100.00% to **5.81%** while clean accuracy is fully preserved at 100.00% (0.51 s). This result matches the ASR reduction quality of the best backdoor benchmarks and could only be discovered through systematic design-space exploration—prior work focusing exclusively on incoming or outgoing directions would have missed it entirely. On ResNet18, the same bias direction with N_i^{Δ} (Δ=1.0, $|\mathcal{D}|$=3) reduces ASR from 85.51% to 40.42% while preserving accuracy (89.49%→89.02%, 2.6 s). Although the residual ASR on ResNet18 remains higher than on MNIST-6, this represents a substantial reduction and indicates that bias-direction repair generalizes beyond small models; the remaining gap likely reflects the deeper, more distributed nature of PGD-style adversarial faults in ResNet18 compared to MNIST-6, where the fault appears more localized to the bias parameters of a single dense layer.

4.3 Effect of Repair Mask Direction and Selection

RQ2. We fix analysis on N (normal) with $|\mathcal{D}|$=5 and vary weight direction and selection policy. Table 4 reports Acc, ASR, repair-weight count (#RepairW), and total wall-clock time.

Direction and mask size jointly determine repair feasibility.

MNIST-BD. Direction is decisive: `outgoing/p95` reduces ASR from 98.68% to **10.98%** with only #RepairW = 6 and 108 s wall time, while all `incoming` variants leave ASR above 87% and collapse clean accuracy (47–50%), despite much larger masks (300–474 repair weights). `Outgoing/r-50` increases #RepairW to 60 but still yields substantially worse ASR (82.32%), showing that mask size alone does not determine repair effectiveness.

GTSRB. No direction achieves meaningful ASR reduction: `out/p95` preserves clean accuracy (96.49%) but leaves ASR at 90.90%, while `incoming/p95` and

Table 4 Direction and selection comparison (N/normal, attack-only, $|\mathcal{D}|$=5). #RepairW = ones in the boolean repair mask. ASR = 100% − repair_rate.

Model	Dir / Sel	k	#RepairW	Time (s)	Acc / ASR (%)
MNIST BD	bias	–	6	173	99.57 / 88.66
	in / p95	p95	474	153	47.36 / 87.22
	in / r-50	50	300	172	49.92 / 89.18
	out / p95	p95	6	108	**99.56 / 10.98**
	out / r-50	50	60	121	47.17 / 82.32
GTSRB	in / p95	p95	52	852	0.48 / 99.52
	in / r-50	50	100	858	35.54 / 95.42
	out / p95	p95	6	803	96.49 / 90.90

Table 5 Analysis-network comparison (incoming/r-50, attack-only, $|\mathcal{D}|$=5; best Δ per surrogate). #Assert = normal + attack asserts.

Model	Formulation	Δ	Acc (%)	ASR (%)	#Assert	Time (s)
Fashion	normal	0.0	93.36	6.64	58	255
	weight-δ	1.5	93.36	6.64	66	294
	input-δ	2.0	88.83	11.17	80	246
MNIST-BD	normal	0.0	49.92	89.18	1,750	172
	weight-δ	2.0	91.31	88.31	3,964	184
	input-δ	1.5	**99.14**	88.16	**485**	175
CIFAR-10	normal	0.0	63.67	88.75	730	512
	weight-δ	2.5	65.10	87.55	2,482	611
	input-δ	2.0	**87.60**	88.49	**398**	606
MNIST-6	normal	0.0	74.42	88.37	163	2
	weight-δ	1.5	**100.0**	89.53	163	2
	input-δ	1.5	95.35	88.37	343	2

`incoming/r-50` both collapse clean accuracy (0.48% and 35.54%, respectively). This confirms the current scalability boundary of single-layer repair on this benchmark.

Bias as a tractability probe.

`Bias` uses the smallest repair mask on MNIST-BD (#RepairW = 6) and serves as a tractable diagnostic baseline. Because it is structurally incapable of redirecting incoming attack activations, we recommend running bias repair first to confirm solver budget and localization stability before committing to outgoing or incoming with appropriate Δ calibration.

4.4 Effect of Ternary and Gated Surrogates

RQ3. We fix the repair mask to incoming/r-50 with $|\mathcal{D}|$=5 and compare N, N_w^Δ (ternary), and N_i^Δ (activation-gated). Table 5 reports the best Δ per surrogate.

Surrogate choice primarily governs *accuracy preservation*, not ASR level, when the mask direction is incoming. On MNIST-BD and CIFAR-10, all three networks leave ASR above 87% because incoming/r-50 is an ineffective repair direction for these models (RQ2). However, concolic execution on N collapses MNIST-BD accuracy to 49.92% (1,750 asserts), while N_i^Δ recovers accuracy to **99.14%** with only 485 asserts. The same pattern holds on CIFAR-10: N_i^Δ achieves the highest accuracy (87.60%) with the fewest asserts (398) compared to N (63.67%, 730 asserts) and

Table 6 Effect of Δ under N_w^Δ on MNIST-6 (incoming/r-50, $|\mathcal{D}|$=5).

Δ	#Assert ↓	Acc (%)	ASR (%)	Total wall-clock time (s)
1.0	263	88.37	89.53	1.97
1.5	163	**100.0**	89.53	1.69
2.0	36	90.70	89.53	1.57
2.5	36	90.70	89.53	1.58
3.0	36	48.84	**95.35**	1.57

N_w^Δ (65.10%, 2,482 asserts). On Fashion-MNIST (a small tractable model), surrogate choice has minimal effect when the direction is already optimal. On MNIST-6, however, surrogate and direction interact critically: all three networks under incoming/r-50 leave ASR near 89%, while switching to N_i^Δ with the `bias` direction reduces ASR to 5.81%—a result achievable only through joint direction–surrogate co-selection. These results show that *surrogate and direction must be co-designed*: incoming/r-50 with any network fails on MNIST-BD, whereas outgoing/p95 with N_i^Δ succeeds (RQ1, Table 3); and for adversarial repair, the bias direction unlocks the fault space that incoming and outgoing directions cannot reach.

4.5 Effect of Abstraction Strength

RQ4. We study the effect of Δ on MNIST-6 under a *fixed direction* (N_w^Δ, incoming/r-50, $|\mathcal{D}|$=5) to isolate the Δ dimension from direction effects (Table 6). As shown in RQ1, switching to bias-direction N_i^Δ achieves ASR 5.81%; the purpose of this analysis is to characterize how Δ modulates assert count and accuracy fidelity when direction is held constant.

As Δ increases, assert count falls monotonically (263→163→36) and total wall-clock time decreases accordingly (1.97 s→1.57 s). Repair quality, however, is *non-monotonic*: Δ=1.5 achieves perfect clean accuracy (100.0%) despite more asserts than $\Delta \geq 2$, while Δ=3.0 collapses accuracy to 48.84% and worsens ASR to 95.35%. Crucially, ASR remains effectively unchanged across all Δ values under this direction (≈89–95%), confirming that Δ modulates fidelity to clean execution paths but cannot compensate for a suboptimal direction choice. The contrast with the RQ1 result (ASR 5.81% under bias+N_i^Δ) underscores that *direction selection is a prerequisite for adversarial repair*: no amount of Δ tuning can substitute for choosing a direction that covers the adversarial fault locus.

4.6 Comparison with CARE

We compare against CARE Sun et al. (2022), a representative grey-box repair method that performs causality-based neuron localization followed by Particle Swarm Optimization (PSO) to adjust identified weights. To compare repair mechanisms under aligned localization coverage, both methods localize the top 1% of critical neurons before repair. CARE's reported results are 5-run averages on the held-out test split

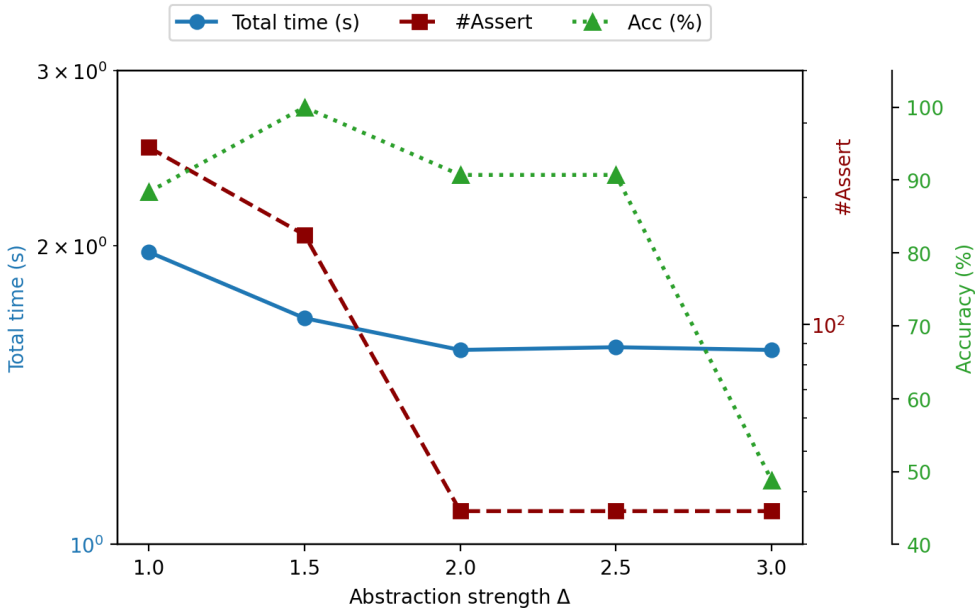


Fig. 6 Total wall-clock time, #Assert, and accuracy under different Δ values on MNIST-6 (N_w^{Δ}, incoming/r-50, $|\mathcal{D}|$=5; log scale for total time and #Assert).

Table 7 Comparison with CARE on backdoor benchmarks (both methods localize top 1% critical neurons). Before values are shared baselines (ours); CARE After values are 5-run averages on test [1000:2000]; Ours are retrospectively best explored $\mathcal{C}^*$.

Model	Before ASR	After ASR (%)		Before Acc	After Acc (%)		Time (s)	
		CARE	**Ours**		CARE	**Ours**	CARE	**Ours**
MNIST-BD	98.68	36.10	**7.05**	99.09	98.40	**99.29**	8,745	**74.4**
Fashion	99.77	36.20	**6.66**	91.36	91.50	**93.34**	681	**160.4**
CIFAR-10	88.34	71.40	**68.75**	97.70	71.60	**95.65**	901	408.2
GTSRB	90.48	**0.00**	85.96	96.51	97.50	92.89	1,428	694.2

[1000:2000], whereas our results correspond to the retrospectively best explored configuration $\mathcal{C}^*$ as defined in Section 4.2.[1] Table 7 reports the comparison. We use our own before-repair baseline values throughout, as the repair targets are the same backdoor-injected models.

MNIST-BD and Fashion-MNIST.

Under the aligned top-1% neuron localization setting and the protocol differences noted above, our retrospectively best explored configurations achieve lower post-repair ASR on both benchmarks: 7.05% vs. CARE's 36.10% average on MNIST-BD and 6.66% vs. 36.20% on Fashion-MNIST. CARE preserves clean accuracy reasonably well

[1]CARE averages over five independent PSO runs, while we report the single best-configuration result from a design-space sweep; the exact test-split boundaries may also differ. These comparisons should be read as evidence of repair potential rather than strict head-to-head dominance claims.

under this setting (MNIST-BD: 98.40%, a 0.69 pp drop; Fashion: 91.50%, a slight improvement), but still leaves over one-third of backdoor-triggered inputs misclassified. Our best explored configurations further reduce ASR while preserving or improving clean accuracy (99.29% and 93.34%, respectively). Wall-clock time also favors our pipeline on these two benchmarks (74.4 s vs. CARE's 8,745 s on MNIST-BD; 160.4 s vs. 681 s on Fashion), reflecting the difference between one-shot Max-SMT over collected path constraints and iterative PSO over full-dataset forward passes.

CIFAR-10.

On CIFAR-10, our best explored configuration achieves modestly lower post-repair ASR than CARE's average (68.75% vs. 71.40%) while preserving substantially more clean accuracy (95.65% vs. 71.60%, a 26.1 pp gap). CARE's PSO search drives down ASR only at the cost of catastrophic accuracy loss on this benchmark, whereas our path-constraint formulation limits the accuracy drop to 2.05 pp under the explored setting. Wall-clock time is also favorable for our method (408.2 s vs. 901 s). This result suggests that, once localization coverage is held constant, symbolic path-constraint repair can match or exceed heuristic PSO on ASR while better preserving clean-input behavior—subject to the reporting-protocol caveats above.

GTSRB.

GTSRB represents the clearest win for CARE: under the aligned top-1% neuron setting, CARE achieves complete backdoor removal (ASR 90.48% $\rightarrow$ 0.00%) while slightly improving clean accuracy to 97.50%, in 1,428 s. Our best explored configuration, by contrast, reduces ASR only marginally (90.48%$\rightarrow$85.96%) at a cost of 3.62 pp accuracy drop (92.89%, 694.2 s). This outcome is consistent with the scalability analysis in Section 4.3: on GTSRB's 512-unit dense layer, no direction or selection policy we explored achieves meaningful ASR reduction under the single-layer repair budget. The CARE result confirms that the GTSRB backdoor is in principle repairable under the same localization budget; the gap therefore reflects a limitation of our current repair scope rather than an inherent property of the model.

5 Discussion

5.1 Constraint Count as the Primary Cost Driver

Our results reveal a counter-intuitive finding: wall-clock time is governed by constraint count rather than model size. GTSRB, one of the larger benchmarks, requires up to 858 s for incoming/r-50 (#RepairW = 100), while MNIST-6—a far smaller model—completes all runs in under 2 s regardless of surrogate choice. This decoupling has an important practical implication: the choice of analysis network and Δ is more consequential than model architecture for repair tractability, and calibrating these parameters per model is a necessary step before deployment.

More broadly, scalability bottlenecks stem from the symbolic complexity of the repair sub-network under the chosen surrogate. N_i^{Δ} reduces assert count most aggressively among the three networks (e.g., 485 asserts on MNIST-BD vs. 1,750 for N

and 3,964 for N_w^{Δ}), while the outgoing/p95 direction dramatically reduces repair-mask size independent of surrogate (e.g., #RepairW = 6 vs. 300–474 for incoming on MNIST-BD).

5.2 Design-Space Implications

Our exploratory methodology suggests practical guidance: (i) start from N with bias direction to probe solver budget; (ii) switch to outgoing/p95 for backdoor benchmarks—this is the most effective direction on MNIST-BD and Fashion-MNIST while producing the smallest repair masks; (iii) apply N_i^{Δ} on dense-heavy models where concolic execution on N collapses accuracy; (iv) calibrate Δ per model to balance fidelity and tractability; (v) always rank candidates by minimizing ASR subject to an accuracy drop bound. We do not claim a universally dominant $\mathcal{C}^*$; rather, surrogate choice, direction, and Δ interact with model scale, layer geometry, and fault type.

5.3 Threats to Validity

We explore 4,737 runs across six benchmarks but report retrospectively best $\mathcal{C}^*$ per benchmark without Bonferroni correction; headline results therefore characterize repair *potential* under our exploration budget rather than establishing strict superiority over any fixed baseline. Because $\mathcal{C}^*$ is chosen after sweeping many configurations using the same evaluation set $\mathcal{T}$, Table 3 should be read as design-space characterization rather than a fully blind deployment protocol; future work should hold out a dedicated validation split for configuration selection and report mean/std over repeated seeds.

Comparison with CARE.

CARE reports 5-run PSO averages, whereas we report the single best explored configuration from a much larger design-space sweep. This asymmetry makes direct numerical dominance claims inappropriate; the CARE comparison in Section 4.6 is intended to illustrate complementary strengths (e.g., CARE's complete GTSRB removal) rather than to establish statistically controlled superiority.

Data partitioning.

All repair-time stages (DeepSHAP localization, constraint collection, and Top-K candidate ranking) use only the repair pool $\mathcal{P}$; reported Acc/ASR use the disjoint evaluation set $\mathcal{T}$. ResNet18 coverage includes 224 complete runs across bias and incoming directions; the best explored repair (bias direction, N_i^{Δ}, Δ=1.0) achieves substantial ASR reduction (85.51% $\rightarrow$ 40.42%) while preserving clean accuracy.

5.4 Path-Level vs. Point-Wise Repair

A fundamental distinction between our approach and prior constraint-based repair methods such as NNrepair Usman et al. (2021) and PRDNN Sotoudeh and Thakur (2021) is the *level* at which behavioral specifications are encoded. Point-wise methods

encode correctness as a requirement that specific input–output pairs produce correct labels; our approach instead encodes *execution path consistency*—the requirement that internal branching decisions on normal inputs remain unchanged, and that adversarial inputs are steered onto different execution paths.

This path-level specification has two practical advantages. First, a single path constraint captures behavioral consistency across a *region* of the input space (all inputs that would follow the same execution path), not just a single point, providing implicit generalization beyond the collected samples. Second, path-level soft constraints for adversarial inputs specify not just "produce a different output" but "take a different internal route"—a more precise behavioral deviation that leaves fewer degrees of freedom for the solver and reduces the risk of spurious repairs.

5.5 Limitations and Future Directions

Our framework achieves near-complete backdoor removal on Fashion-MNIST (ASR: 99.77% $\rightarrow$ 6.66%) and MNIST-BD (ASR: 98.68% $\rightarrow$ 7.05%), and near-complete adversarial repair on MNIST-6 (ASR: 100.00% $\rightarrow$ 5.81%) when bias-direction N_i^{Δ} is selected. ResNet18 shows partial adversarial reduction (ASR: 85.51% $\rightarrow$ 40.42%), while CIFAR-10 and GTSRB show limited improvement. The following limitations help explain the remaining gaps and suggest directions for future work.

Scalability to large dense layers.

GTSRB and CIFAR-10 illustrate the current scalability ceiling. GTSRB (512-unit dense layer) achieves at best ASR = 85.96% after repair, while CIFAR-10 reaches only 68.75%. Incoming direction collapses accuracy entirely on GTSRB (0.48%), and outgoing/p95—though preserving accuracy—produces no meaningful ASR reduction. Partial symbolic unrolling or hierarchical constraint decomposition may help push past this boundary.

Single-layer, single-direction symbolic repair.

Our Max-SMT instantiation confines repair unknowns to one layer and one weight direction (incoming, outgoing, or bias) per run, which keeps collected constraints in QF_LRA along fixed ReLU paths. Extending repair unknowns across multiple layers—with any per-layer direction—introduces bilinear terms (e.g., $w^{(\ell+1)} \cdot \mathrm{ReLU}(w^{(\ell)}x)$) that fall outside QF_LRA; mixing multiple directions within the same layer would have the same effect. Alternative backends (e.g., MIQP Majd et al. (2021) or CEGIS over non-linear theories) are left to future work.

No formal guarantee.

Path constraints may be collected on analysis surrogates N_w^{Δ} or N_i^{Δ}; faithful validation on N compensates empirically but does not close the surrogate–deployment gap. Abstraction-refinement loops, as studied in verification-guided repair Ma et al. (2024) and counterexample-guided DNN verification Liu et al. (2024), are promising directions for closing this gap automatically.

Input sensitivity.

Repair quality is sensitive to the representativeness of the constraint-collection inputs. An unrepresentative adversarial sample that does not reliably trigger the target fault will produce soft constraints that do not cover the relevant execution paths. Integrating automated input selection—for example, using SHAP-guided sampling to identify inputs that maximally activate the attack-only neuron set S—could reduce this dependence.

Weight selection heuristic.

Random-k budgets trade theoretical optimality for fixed SMT variable counts; percentile-based masks in the artifact may help on some models but did not rescue MNIST 2 in our sweeps.

6 Conclusion

We presented a configurable, constraint-guided framework for post-hoc neural network repair that couples DeepSHAP localization, multi-dimensional repair masks (incoming, outgoing, bias; random-k and percentile selection), path-level concolic specifications on N or analysis surrogates (N_w^Δ, N_i^Δ), and Top-K Max-SMT with mandatory faithful validation on N. By empirically navigating this design space across 4,737 runs on six benchmarks, we show that repair success depends critically on direction–surrogate–Δ co-calibration. On backdoor benchmarks, near-complete removal is achievable on Fashion-MNIST and MNIST-BD (ASR >98%→<8%) when outgoing direction with N_i^Δ is selected among explored configurations. On adversarial benchmarks, bias-direction N_i^Δ repair achieves near-complete adversarial removal on MNIST-6 (ASR 100%→5.81%, 0.51 s) and substantial reduction on ResNet18 (ASR 85.51%→40.42%)—results that would be invisible to any search restricted to incoming or outgoing directions. CIFAR-10 and GTSRB remain challenging under single-layer repair, exposing a tractability–fidelity boundary. The results highlight that systematic design-space exploration, rather than a fixed repair recipe, is necessary for discovering effective repair configurations, and provide a reproducible basis for further exploration of repair dimensions.

Funding

This work was partially supported by the National Science and Technology Council (NSTC), Taiwan, under Grant No. 113-2221-E-004-010-MY2 and 114-2634-F-004-002-MBK.

Competing interests

The authors have no relevant financial or non-financial interests to disclose.

Data availability

The benchmark datasets used in this study (Fashion-MNIST, MNIST, CIFAR-10, and GTSRB) are publicly available from their original sources; backdoored model

checkpoints follow the CARE benchmark setup described in the paper. The repair pipeline source code, concolic constraint-collection scripts, experiment configuration files, and reproduction instructions are publicly available at https://github.com/folivora-hi/Repair. Because trained models, attack samples, precomputed DeepSHAP repair masks, and verification datasets are too large for the repository, they are provided separately via the Google Drive link documented in the repository README (https://github.com/folivora-hi/Repair#data); after download, files should be placed under the `Data/` directory layout described there. Experimental logs and aggregated results reported in the paper are included in the repository or reproducible from the provided scripts and configuration files.

Author contributions

T.-Y. Liu: conceptualization, methodology, software, validation, formal analysis, investigation, data curation, visualization, writing—original draft and review and editing.

F. Yu: supervision, project administration, writing—review and editing.

Acknowledgements

The authors thank Wei-Yu Shi for his earlier exploration of ternary network simplification in concolic testing for neural networks, which inspired the ternary network formulation N_w^Δ and activation-gated surrogate N_i^Δ in this work.

References

Angelova, A., Krizhevsky, A., Vanhoucke, V., Ogale, A.S., Ferguson, D.: Real-time pedestrian detection with deep network cascades. In: Proceedings of the British Machine Vision Conference (BMVC) (2015)

Akhtar, N., Mian, A.: Threat of adversarial attacks on deep learning in computer vision: A survey. IEEE access : practical innovations, open solutions **6**, 14410–14430 (2018) https://doi.org/10.1109/ACCESS.2018.2807555

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., Mané, D.: Concrete Problems in AI Safety (2016). https://doi.org/10.48550/arXiv.1606.06565

Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. In: Proceedings of the International Conference on Learning Representations (ICLR) (2015). https://doi.org/10.48550/arXiv.1409.0473

Bjørner, N., Fazekas, K.: On incremental pre-processing for SMT. In: Automated Deduction – CADE 29, pp. 41–60. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-38499-8_3

Chi, Z., Ma, J., Yang, P., Huang, C.-C., Li, R., Huang, X., Zhang, L.: Patch Synthesis for Property Repair of Deep Neural Networks. https://doi.org/10.48550/arXiv.2404.01642 . http://arxiv.org/abs/2404.01642

Chen, Y.-F., Tsai, W.-L., Wu, W.-C., Yen, D.-D., Yu, F.: PyCT: A Python concolic tester. In: Oh, H. (ed.) Programming Languages and Systems, pp. 38–46. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-89159-9_3

Chen, Z., Zhou, J., Sun, Y., Wang, J., Xuan, Q., Yang, X.: Interpretability Based Neural Network Repair. In: Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing And Analysis. ISSTA 2024, pp. 908–919. Association for Computing Machinery, Vienna, Austria (2024). https://doi.org/10.1145/3650212.3680330 . https://dl.acm.org/doi/10.1145/3650212.3680330

Fu, F., Li, W.: REASSURE: Sound and Complete Neural Network Repair with Minimality and Locality Guarantees. https://doi.org/10.48550/arXiv.2110.07682 . http://arxiv.org/abs/2110.07682

Gu, T., Liu, K., Dolan-Gavitt, B., Garg, S.: Badnets: Evaluating backdooring attacks on deep neural networks. Ieee Access **7**, 47230–47244 (2019) https://doi.org/10.1109/ACCESS.2019.906906

Graves, A., Mohamed, A.-r., Hinton, G.: Speech recognition with deep recurrent neural networks. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 6645–6649. IEEE, Vancouver, BC, Canada (2013). https://doi.org/10.1109/ICASSP.2013.6638947

Gulshan, V., Peng, L., Coram, M., Stumpe, M.C., Wu, D., Narayanaswamy, A., Venugopalan, S., Widner, K., Madams, T., Cuadros, J., *et al.*: Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. JAMA : the journal of the American Medical Association **316**(22), 2402–2410 (2016) https://doi.org/10.1001/jama.2016.17216

Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and Harnessing Adversarial Examples (2014). https://doi.org/10.48550/arXiv.1412.6572

Liu, F., Du, X., Ding, H., Qian, J.: Towards robust neural networks: Exploring counterfactual causality-based repair. Expert Systems With Applications **257**, 125082 (2024) https://doi.org/10.1016/j.eswa.2024.125082

Le Goues, C., Nguyen, T., Forrest, S., Weimer, W.: GenProg: A generic method for automatic software repair. IEEE Transactions on Software Engineering **38**(1), 54–72 (2012) https://doi.org/10.1109/TSE.2011.104

Lundberg, S.M., Lee, S.-I.: A unified approach to interpreting model predictions. In: Advances in Neural Information Processing Systems, vol. 30, pp. 4765–4774 (2017)

Li, F., Liu, B., Wang, X., Zhang, B., Yan, J.: Ternary Weight Networks. https://doi.org/10.48550/arXiv.1605.04711 . http://arxiv.org/abs/1605.04711

Liu, J., Xing, Y., Shi, X., Song, F., Xu, Z., Ming, Z.: Abstraction and refinement: Towards scalable and exact verification of neural networks. ACM Transactions on Software Engineering and Methodology **33**(5), 1–35 (2024) https://doi.org/10.1145/3644387

Lin, Y.-C., Yu, F.: DeepSHAP summary for adversarial example detection. In: 2023 IEEE/ACM International Workshop on Deep Learning for Testing and Testing for Deep Learning (DeepTest), pp. 17–24. IEEE, San Francisco, CA, USA (2023). https://doi.org/10.1109/DeepTest59050.2023.00012

Mellempudi, N., Kundu, A., Mudigere, D., Das, D., Kaul, B., Dubey, P.: Ternary Neural Networks with Fine-Grained Quantization. https://doi.org/10.48550/arXiv.1705.01462 . http://arxiv.org/abs/1705.01462

Ma, J., Wang, J., Xuan, Q., Wang, Z.: ProRepair: Provable Repair of Deep Neural Network Defects by Preimage Synthesis and Property Refinement. In: Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security. CCS '25, pp. 4169–4183. Association for Computing Machinery, Salt Lake City, UT, USA (2025). https://doi.org/10.1145/3719027.3765057 . https://dl.acm.org/doi/10.1145/3719027.3765057

Ma, J., Wang, J., Xuan, Q., Wang, Z.: Provable Fairness Repair for Deep Neural Networks. In: 2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 508–520 (2025). https://doi.org/10.1109/ASE63991.2025.00049 . https://ieeexplore.ieee.org/abstract/document/11334285

Mechtaev, S., Yi, J., Roychoudhury, A.: DirectFix: Looking for simple program repairs. In: Proceedings of the 37th IEEE/ACM International Conference on Software Engineering, pp. 448–458. IEEE, Florence, Italy (2015). https://doi.org/10.1109/ICSE.2015.63

Mechtaev, S., Yi, J., Roychoudhury, A.: Angelix: Scalable multiline program patch synthesis via symbolic analysis. In: Proceedings of the 38th International Conference on Software Engineering, pp. 691–701. ACM, Austin, TX, USA (2016). https://doi.org/10.1145/2884781.2884807

Ma, J., Yang, P., Wang, J., Sun, Y., Huang, C.-C., Wang, Z.: VeRe: Verification Guided Synthesis for Repairing Deep Neural Networks. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. ICSE '24, pp. 1–13. Association for Computing Machinery, Lisbon, Portugal (2024). https://doi.org/10.1145/3597503.3623332 . https://dl.acm.org/doi/10.1145/3597503.3623332

Majd, K., Zhou, S., Amor, H.B., Fainekos, G., Sankaranarayanan, S.: Local Repair of Neural Networks Using Optimization. https://doi.org/10.48550/arXiv.2109.14041 .

http://arxiv.org/abs/2109.14041

Nguyen, H.D.T., Qi, D., Roychoudhury, A., Chandra, S.: SemFix: Program repair via semantic analysis. In: Proceedings of the 35th International Conference on Software Engineering, pp. 772–781. IEEE, San Francisco, CA, USA (2013). https://doi.org/10.1109/ICSE.2013.6606623

Nightingale, P., Spracklen, P., Miguel, I.: Automatically improving SAT encoding of constraint problems through common subexpression elimination in Savile Row. In: Principles and Practice of Constraint Programming, pp. 330–340. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23219-6_23

Strubell, E., Ganesh, A., McCallum, A.: Energy and policy considerations for deep learning in NLP. In: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL), pp. 3645–3650 (2019). https://doi.org/10.18653/v1/P19-1355

Sun, Y., Huang, X., Kroening, D., Sharp, J., Hill, M., Ashmore, R.: DeepConcolic: Testing and Debugging Deep Neural Networks. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), pp. 111–114 (2019). https://doi.org/10.1109/ICSE-Companion.2019.00051 . https://ieeexplore.ieee.org/abstract/document/8802786

Sun, X., Liu, W., Wang, S., Chen, T., Tao, Y., Mao, X.: AutoRIC: Automated Neural Network Repairing Based on Constrained Optimization. ACM Trans. Softw. Eng. Methodol. **34**(2), 50–15029 (2025) https://doi.org/10.1145/3690634

Sun, B., Sun, J., Pham, H.L., Shi, J.: CARE: Causality-based Neural Network Repair. https://doi.org/10.48550/arXiv.2204.09274 . http://arxiv.org/abs/2204.09274

Sotoudeh, M., Thakur, A.V.: PRDNN: Provable repair of deep neural networks. In: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. PLDI 2021, pp. 588–603. Association for Computing Machinery, Virtual Event (2021). https://doi.org/10.1145/3453483.3454064 . https://dl.acm.org/doi/10.1145/3453483.3454064

Sun, Y., Wu, M., Ruan, W., Huang, X., Kwiatkowska, M., Kroening, D.: Concolic testing for deep neural networks. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pp. 109–119 (2018). https://doi.org/10.1145/3238147.3238187

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing Properties of Neural Networks (2013). https://doi.org/10.48550/arXiv.1312.6199

Tao, Z., Nawas, S., Mitchell, J., Thakur, A.V.: APRNN: Architecture-Preserving Provable Repair of Deep Neural Networks. Reproduction Package for the PLDI

2023 Article "Architecture-Preserving Provable Repair of Deep Neural Networks" **7**, 124–443124467 (2023) https://doi.org/10.1145/3591238

Usman, M., Gopinath, D., Sun, Y., Noller, Y., Pasareanu, C.: NNrepair: Constraint-based Repair of Neural Network Classifiers. https://doi.org/10.48550/arXiv.2103.12535 . http://arxiv.org/abs/2103.12535

Vares, F., Johnson, B.: Causality-driven neural network repair: Challenges and opportunities. In: Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering, pp. 1406–1409. ACM, Trondheim, Norway (2025). https://doi.org/10.1145/3696630.3731615

Yang, A., Bergsträßer, P., Zetzsche, G., Chiang, D., Lin, A.W.: Length generalization bounds for transformers. arXiv preprint arXiv:2603.02238 (2026) https://doi.org/10.48550/arXiv.2603.02238

Yuviler, T., Drachsler-Cohen, D.: Enhancing neural network robustness via synthesis of repair programs. In: Static Analysis: 32nd International Symposium (SAS 2025), pp. 221–248. Springer, Berlin, Heidelberg (2025). https://doi.org/10.1007/978-3-032-07106-4_10