REPTRAN: Search-Based Repair of Transformer Models

Yuta Ishimoto*, Paolo Arcaini†, Fuyuki Ishikawa†, Masanari Kondo‡, Naoyasu Ubayashi§, and Yasutaka Kamei‡
*The University of Osaka, Japan, †National Institute of Informatics, Japan,
‡Kyushu University, Japan, §Waseda University, Japan

***Abstract*—To ensure the overall quality of AI-enabled software, not only traditional software components but also AI components need to be tested and repaired. Among AI components, Transformer models are increasingly integrated into software systems, which makes their misbehaviors critical. Although prior work in the software engineering community has proposed deep neural network (DNN) repair methods, most overlook Transformer-specific structures. We propose REPTRAN, a search-based repair method for Transformer models. It targets their feed-forward networks (FFNs), which play a central role in the architecture. REPTRAN identifies suspicious weights by combining two types of scores: a variance-based neuron score and an existing bidirectional score. It then iteratively optimizes these weights using differential evolution. Our evaluation includes 18 fault benchmarks constructed from CIFAR-100 and Tiny-ImageNet. We compare REPTRAN against three baselines: random weight selection, ARACHNE (a state-of-the-art DNN repair method), and ARACHNEW, which enables ARACHNE to control the number of selected weights. REPTRAN achieved an average repair rate of 74.7%, statistically outperforming random selection and ARACHNE across all benchmarks. Effect size analysis revealed that REPTRAN achieved higher repair rates than ARACHNEW regardless of the number of selected weights. These results suggest that REPTRAN is effective for enhancing the reliability of AI-enabled software.**

***Index Terms*—AI-enabled software, Transformer, Repair, Reliability**

I. INTRODUCTION

The Transformer [1] has become a critical neural network architecture underlying foundation models such as large language models (LLMs). Since Transformer models are increasingly used in a wide range of AI-enabled software (e.g., autonomous driving [2]), misbehaviors of these models can significantly compromise the reliability of the entire system. Indeed, the OECD AI Incidents and Hazards Monitor has documented numerous cases of failures in autonomous driving systems [3], many of which resulted in severe injuries, fatalities, and significant financial losses.

One promising direction for addressing such misbehaviors is *deep neural network (DNN) repair* [4]–[11]. These methods are designed to complement full retraining: while full retraining aims to improve overall performance (e.g., accuracy), DNN repair targets specific types of misbehavior. It identifies and updates only the components of the model (e.g., neurons or weights) that are responsible for the misbehavior. This selective update seeks to correct the targeted misbehavior while preserving the existing correct behaviors.

Given this background, there is a growing need for repair methods tailored to Transformer models; however, most existing DNN repair methods have focused on conventional networks (e.g., convolutional neural networks (CNNs) [5]–[7], [9]), with limited attention to Transformer models [10]. Unlike such conventional networks, Transformer models rely on unique architectural components such as multi-head self-attention (MSA) and feed-forward networks (FFNs) consisting of two fully connected layers with a non-linear activation in between [1]. Existing methods do not account for these components, indicating the need for repair methods designed for Transformer models.

To this end, we propose REPTRAN, a novel repair approach for Transformer models. To identify which weights should be modified, REPTRAN computes a *weight suspiciousness score*, which reflects the contribution of each weight to the misbehavior. This score consists of two components: (1) a *neuron score* and (2) a *bidirectional score*. The neuron score, newly introduced in this work, is based on the mean and variance of activations of intermediate FFN neurons. The bidirectional score comes from ARACHNE [5], a state-of-the-art DNN repair method. Weights with high suspiciousness scores are selected for repair. These weights are optimized using differential evolution (DE) [12], a population-based search algorithm. The fitness function is designed to maximize the number of repaired misbehaviors of the target type while minimizing disruptions to the existing correct behaviors.

We evaluated REPTRAN on 18 fault benchmarks constructed from two image classification datasets: CIFAR-100 [13] and Tiny-ImageNet [14]. We used the Vision Transformer (ViT) [15] as the target model in our experiments. REPTRAN achieved an average repair rate of 74.7% across the 18 fault benchmarks, reaching up to 95.2% in the best case. In contrast, applying the original ARACHNE to the FFNs of ViT resulted in a much lower average repair rate of 17.1%, with a maximum of 44.4%. REPTRAN also demonstrated competitive efficiency, with an average runtime of 476.19 seconds, which was faster than ARACHNE (620.93 seconds on average).

The contributions of this study are as follows:

- We propose REPTRAN, a novel DNN repair method tailored to Transformer models.
- We develop a weight selection strategy based on a *neuron score* that quantifies behavioral differences in intermediate FFN neurons in the Transformer.

- Our evaluation on 18 fault benchmarks demonstrates that REPTRAN outperforms ARACHNE and random baselines in both repair rate and execution time.
- All source code for our experiments is publicly available.[1]

II. BACKGROUND AND RELATED WORK

In this section, we begin by introducing existing DNN repair methods. Next, we describe the Transformer architecture to provide background for our work. Finally, we introduce existing work on internal analyses and editing of Transformer models that motivate our proposed repair method, REPTRAN.

A. DNN Repair

The goal of DNN repair [4]–[11] is to achieve controlled modifications (e.g., fixing specific types of misclassifications). Full retraining and repair serve complementary roles: while full retraining pursues overall performance improvement when sufficient time and data are available, repair enables more controlled improvements even when such resources are limited.

Among DNN repair methods, search-based approaches are common [4]–[7], [16]. They typically involve two phases: (1) a *localization* phase that identifies the neurons or weights to be repaired, and (2) a *search* phase that uses metaheuristics to optimize parameter modifications. For instance, ARACHNE [5] considers both forward impact and gradient loss to identify faulty weights. Forward impact measures the influence of the weight on the final output, while gradient loss measures its contribution to misbehaviors. Faulty weights are identified as the Pareto front of these two metrics, and DE [12] is applied to modify their values. Meanwhile, CARE [4] identifies faulty neurons by measuring the causal effect of each neuron on the output. The identified neurons are then refined using particle swarm optimization (PSO) [17]. Our method, REPTRAN, follows the search-based approach but differs from existing methods in that it targets Transformer models and introduces a novel weight selection strategy for their FFNs.

Another approach is to formulate the entire DNN repair process as an optimization problem (e.g., linear or quadratic programming) and then rely on a solver to produce repair results [8]–[10], [18]. For instance, PRDNN [18] and APRNN [8] translate the DNN repair problem into an linear programming problem, whereas AutoRIC [9] adopts a quadratic programming problem. More recently, PRoViT [10] was introduced as a provable repair method for ViT, ensuring correct classifications on a designated repair set. PRoViT addresses the scalability issue of solver-based approaches by restricting repair targets to the final layer only. It provides three variants that differ in how this final layer is modified: (1) $\text{PRoViT}_{\text{LP}}$ solves a linear program (LP) over the final layer; (2) $\text{PRoViT}_{\text{FT}}$ fine-tunes (FT) the final layer until it reaches 100% accuracy on the repair set; and (3) $\text{PRoViT}_{\text{FT+LP}}$ first applies one epoch of fine-tuning to the final layer and then runs the above LP, so that the LP repairs whatever the single fine-tuning epoch has not yet fixed.

[1]https://anonymous.4open.science/r/RepTran-replication-8E9E

In our evaluation, we compare REPTRAN against representative methods from both categories: the search-based ARACHNE and the solver-based PRoViT.

B. Transformer

The Transformer [1] consists of multi-head self-attention (MSA) and feed-forward networks (FFN), which are repeatedly stacked in multiple layers. These mechanisms differ significantly from those in CNNs and RNNs, which have been the main focus of DNN repair thus far.

The FFN, which is the primary target of our repair method, consists of two fully connected layers with a non-linear activation function:

$$\text{FFN}(z) = W_{\text{aft}}(\sigma(W_{\text{bef}}z + b_{\text{bef}})) + b_{\text{aft}}, \tag{1}$$

where $z \in \mathbb{R}^D$ denotes the input to the FFN (i.e., the output of the preceding MSA). $W_{\text{bef}} \in \mathbb{R}^{D_{\text{hidden}} \times D}$ and $W_{\text{aft}} \in \mathbb{R}^{D \times D_{\text{hidden}}}$ are learnable weight matrices, while $b_{\text{bef}} \in \mathbb{R}^{D_{\text{hidden}}}$ and $b_{\text{aft}} \in \mathbb{R}^D$ are bias terms. σ represents a non-linear activation function such as ReLU or GELU. Typically, $D_{\text{hidden}} = 4D$; that is, the FFN projects MSA outputs to a higher dimension via the first layer and then back to the original dimension via the second layer. This structure is believed to contribute to the expressiveness of Transformer models [19], [20].

C. Internal Analysis and Editing of Transformer

Internal Analysis. While Transformers have achieved remarkable success across various tasks, their internal mechanisms remain black boxes. Therefore, a lot of studies have attempted to elucidate the functions of key components, such as MSA [21]–[23] and FFN [19], [20]. This section introduces these existing studies; although they emerge from the AI research community, they still provide valuable insights for the development of repair methods for Transformer models.

Michel et al. [21] have observed that a large percentage of attention heads can be removed at test time without significantly decreasing performance. Similarly, Li et al. [23] proposed pruning-based head importance metrics that assess the impact of attention heads both within the attention layers and on the final output for ViTs.

Geva et al. [19] have shown that FFNs in Transformer models operate as key-value memories. Building on this, Dai et al. [20] have shown that the intermediate neurons of FFNs (of dimension D_{hidden}) store learned knowledge, coining the term *knowledge neurons*, with further studies [24], [25] extending this line of research. These findings indicate that FFNs play a crucial role in the Transformer and offer insights relevant to DNN repair. From a repair perspective, identifying knowledge neurons resembles a localization phase. However, knowledge neurons are defined with respect to factual associations rather than the classification misbehaviors we aim to repair; moreover, while knowledge neurons are benign, DNN repair focuses on identifying problematic neurons or weights to be modified. Motivated by these observations, we propose a novel repair method that targets FFNs of Transformer models.

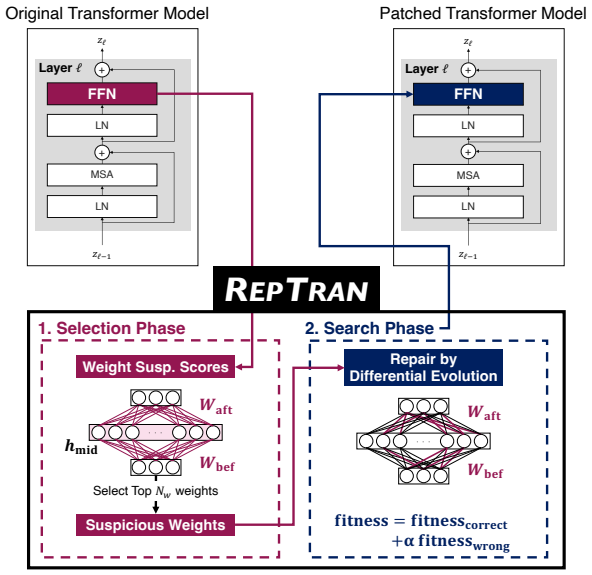


Fig. 1. An overview of REPTRAN.

Model Editing. A related line of work edits Transformer models post hoc to change a specified factual association while preserving unrelated behavior [26], [27]. Both methods take a single edit pair, i.e., a prompt and its new desired output, and write exactly that association into the model. MEND [26] transforms the fine-tuning gradient of the edit pair into a weight update using auxiliary editor networks, whereas ROME [27] directly writes a single key-value association into an FFN located by causal tracing, via a rank-one weight update. In contrast, a repair request is not an edit pair with a known target association but a set of misclassified inputs sharing a misbehavior type, and the patch must generalize to unseen inputs of the same type rather than to paraphrases of one edited prompt. Moreover, ROME presupposes the subject-relation-object structure of factual prompts and MEND requires editor training on an edit dataset, neither of which applies to our setting; the evaluation protocols also differ (paraphrase generalization vs. repair and break rates on held-out inputs). This distinction also explains our choice of baseline: PRoViT is a direct baseline because it targets ViT repair under the same repair and break evaluation, whereas adapting MEND and ROME would require redefining edit pairs and is left as future work.

III. REPTRAN: SEARCH-BASED REPAIR OF TRANSFORMER MODELS

Fig. 1 illustrates the overall process of REPTRAN, our search-based repair method for Transformer models. The input is the *original Transformer model*, and the output is the *patched Transformer model*. REPTRAN consists of two main phases: (1) a *selection* phase, where we compute neuron- and weight-level scores in the FFN to identify suspicious weights; and (2) a *search* phase, where a search algorithm modifies these weights, guided by a fitness function.

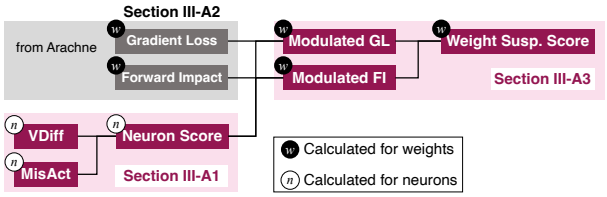


Fig. 2. Composition of the weight suspiciousness score.

A. Selection Phase

The goal of the selection phase is to narrow down the weight parameter space by identifying suspicious weights. We target both W_{bef} and W_{aft} in the FFN (see Eq. 1) for selection. Such feed-forward layers also appear in other architectures such as CNNs and RNNs, and our neuron score could in principle be computed on them as well. We nonetheless focus on the FFNs of Transformers, whose intermediate neurons act as key-value memories that store semantic associations [19], [20]. Our neuron score is inspired by this property, which motivates the positioning of REPTRAN as a Transformer-oriented repair method. Extending REPTRAN to general DNN repair beyond Transformers is technically feasible but left for future work.

For each weight in these matrices, REPTRAN computes a *weight suspiciousness score*, which combines two components: *neuron score* and *bidirectional score*. Fig. 2 shows the composition of the weight suspiciousness score. The neuron score (see Sect. III-A1) quantifies the contribution of intermediate FFN neurons by using the mean and variance of their activations. The bidirectional score (see Sect. III-A2) measures the forward impact and gradient loss of each weight. The weight suspiciousness score is obtained by modulating its bidirectional score with the corresponding neuron score (see Sect. III-A3). The bidirectional score alone does not account for the importance of individual neurons in the FFN of Transformer models; therefore, we explicitly incorporate this aspect into our approach. Finally, this phase outputs suspicious weights, i.e., those with high suspiciousness scores.

1) Neuron Score: As shown at the bottom of Fig. 2, the neuron score quantifies the contribution of intermediate FFN neurons to the misbehavior. Unlike knowledge neurons [20], which rely on integrated gradients [28] requiring multiple forward and backward passes per input, our score is designed specifically for efficient repair by requiring only a single forward pass per input. It consists of two components, *VDiff* and *MisAct*, described below.

VDiff quantifies how differently each intermediate FFN neuron h^i_{mid} behaves between correct and incorrect inputs, using the variance of activations. Motivated by the use of variance as a coverage metric for DNNs [29] and as a risk indicator for model outputs [30], we define VDiff as follows:

$$\text{VDiff}(h^i_{\text{mid}}) = \left| \sum_{j=1}^{D_{\text{hidden}}} |\Sigma^{\text{cor}}_{ij}| - \sum_{j=1}^{D_{\text{hidden}}} |\Sigma^{\text{mis}}_{ij}| \right|,$$

where Σ^{cor} and Σ^{mis} are the covariance matrices of the hidden state $h_{\text{mid}} \in \mathbb{R}^{D_{\text{hidden}}}$, computed from the correctly classified

samples ($\mathcal{I}_{\text{cor}}$) and the misclassified samples ($\mathcal{I}_{\text{mis}}$), respectively. Each summation $\sum_j |\Sigma_{ij}|$ aggregates the neuron's own variance and absolute covariances with all other neurons, capturing the diversity of its behavior under a given input set. VDiff thus identifies neurons whose behavioral diversity differs between correct and incorrect inputs.

MisAct measures the mean activation of neuron h^i_{mid} under the misbehavior:

$$\text{MisAct}(h^i_{\text{mid}}) = \frac{1}{|\mathcal{I}_{\text{mis}}|} \sum_{(x,y)\in\mathcal{I}_{\text{mis}}} a_i(x),$$

where $a_i(x)$ is the activation value of neuron h^i_{mid} for input x. While VDiff captures variability, MisAct reflects how strongly the neuron fires on average under misbehavior.

We define the neuron score for h^i_{mid} as follows:

$$\text{NeuronScore}(h^i_{\text{mid}}) = \text{VDiff}_{\text{norm}}(h^i_{\text{mid}}) \times \text{MisAct}_{\text{norm}}(h^i_{\text{mid}}).$$

Because VDiff and MisAct have different scales, we normalize each of them to $[0, 1]$ (i.e., $\text{VDiff}_{\text{norm}}$ and $\text{MisAct}_{\text{norm}}$), ensuring that the neuron score lies in $[0, 1]$. We use the product of VDiff and MisAct to capture an *and condition*: we emphasize neurons that show both a significant variance-based shift *and* strong activation under misbehavior. This neuron score is combined with the bidirectional score, which is introduced next, to guide the selection of suspicious weights.

2) Bidirectional Score: As shown in the gray parts in the middle of Fig. 2, our selection phase uses the two existing metrics from ARACHNE [5]: *forward impact (FI)*, which measures the influence of a weight on the final output, and *gradient loss (GL)*, which captures the gradient of the loss with respect to the weight.

3) Weight Suspiciousness Score: As shown at the top of Fig. 2, we compute the *weight suspiciousness score* by combining the neuron score and the bidirectional score. For each weight $w \in W_{\text{bef}} \cup W_{\text{aft}}$, there exists exactly one intermediate neuron h^i_{mid} to which w is connected (see the middle of Fig. 1). We modulate the bidirectional scores as follows:

$$\text{ModFI}(w) = \text{FI}(w) \times \text{NeuronScore}(h^i_{\text{mid}}),$$
$$\text{ModGL}(w) = \text{GL}(w) \times \text{NeuronScore}(h^i_{\text{mid}}).$$

These modulated scores are normalized to $[0, 1]$, and the final weight suspiciousness score is:

$$\text{WeightSusp}(w) = p\text{ModFI}(w) + (1-p)\text{ModGL}(w), \quad (2)$$

where p is a hyperparameter that controls the trade-off between the two scores. We select the top-N_w weights with the highest scores as suspicious weights.

B. Search Phase

After selecting suspicious weights, we apply a metaheuristic to repair the misbehavior without harming correct behavior.

By default, REPTRAN employs DE [12], although other search algorithms (e.g., PSO) can also be used. DE maintains a population of candidate solutions, each representing a set of weight values for the selected weights. These candidates are iteratively evolved through mutation and selection over multiple generations, guided by a fitness function. Each candidate is initialized by sampling from a normal distribution whose mean and variance match those of the corresponding weight matrix (W_{bef} or W_{aft}). The search terminates after $G_{\max}$ generations or when the best fitness stagnates for G_{stag} generations.

The fitness function that DE aims to maximize is designed to reduce misbehavior while preserving correct behavior:

$$\text{Fitness}(u) = \sum_{(x,y)\in\mathcal{I}_{\text{cor}}} \frac{S_u(x,y)}{|\mathcal{I}_{\text{cor}}|} + \alpha \sum_{(x,y)\in\mathcal{I}_{\text{mis}}} \frac{S_u(x,y)}{|\mathcal{I}_{\text{mis}}|}, \quad (3)$$

where u is a candidate solution, α is a hyperparameter that balances maintaining correctness and reducing misbehavior, and $S_u(x, y)$ returns 1 if the model with modified weights u correctly predicts the label for input x, and $\frac{1}{1+\mathcal{L}(x,y)}$ otherwise. Since correct predictions receive the maximum $S_u(x, y)$ of 1 while incorrect ones receive a value less than 1, maximizing this function drives DE toward solutions that preserve correct behavior while reducing misbehavior.

IV. EXPERIMENTAL DESIGN

A. Research Questions

We formulate five RQs to evaluate the effectiveness and efficiency of REPTRAN in repairing DNNs, to investigate the impact of key parameters on its performance, and to compare it with a repair method tailored to ViTs.

RQ1 (Effectiveness): How effective is REPTRAN compared with the search-based baselines?

RQ2 (Efficiency): How efficient is REPTRAN compared with the search-based baselines?

RQ3 (Number of Selected Weights): How does the number of selected weights affect the repair performance?

RQ4 (Balance Coefficients): How do the balance coefficients in each phase affect repair performance?

RQ5 (Comparison with a ViT-specific method): How does REPTRAN compare with PRoViT?

B. Datasets

We use two datasets in our experiments: *CIFAR-100 (C100)* [13] and *Tiny-ImageNet (TinyImg)* [14], which are widely used for image classification tasks. C100 is an image classification dataset with 100 classes, consisting of 50,000 training and 10,000 test samples of 32x32 color images. TinyImg is a smaller version of ImageNet with 200 classes, consisting of 100,000 training, 10,000 validation, and 10,000 test samples of 64x64 color images.

To ensure consistency in data splitting, we follow prior studies on DNN repair [5], [31] and adopt a three-way split, i.e., *train*, *repair*, and *test*. The train set is used to fine-tune the model, the repair set to identify and correct misbehavior, and the test set to evaluate how much the targeted misclassification has been reduced on data unseen during repair. For both datasets, we split the original training set into train and repair sets with an 80:20 ratio. For C100, the original test set

TABLE I
NUMBER OF MISCLASSIFIED SAMPLES FOR REPAIR AND TEST SETS.

Type (Rank)	*SRC-TGT* (1/2/3)			*TGT-FP* (1/2/3)			*TGT-FN* (1/2/3)		
C100									
$\lvert\mathcal{I}_{\text{mis}}^{\text{repair}}\rvert$	19	16	14	27	32	31	27	33	27
$\lvert\mathcal{I}_{\text{mis}}^{\text{test}}\rvert$	21	8	8	34	23	42	28	25	29
TinyImg									
$\lvert\mathcal{I}_{\text{mis}}^{\text{repair}}\rvert$	33	20	18	56	46	41	39	32	34
$\lvert\mathcal{I}_{\text{mis}}^{\text{test}}\rvert$	14	12	9	25	15	25	20	13	16

is used as-is, resulting in 40,000/10,000/10,000 samples for train/repair/test. For TinyImg, since the original test set lacks ground-truth labels, we use the original validation set as the test set, resulting in 80,000/20,000/10,000 samples.

C. Transformer Models

We use ViT [15] as the subject model, a representative Transformer architecture widely adopted in vision tasks. We use the publicly available weights on Hugging Face.[2] The model consists of 12 Transformer encoder blocks, each containing a FFN with a hidden dimension of 3,072 (i.e., $D_{\text{hidden}} = 3072, D = 768$). We focus on ViT because vision tasks often involve safety-critical applications such as autonomous driving [2] and medical imaging [32], and an industry report [33] emphasizes the demand for practical DNN repair methods in this domain. Note that REPTRAN can be applied, in principle, to any Transformer architecture that employs FFNs.

In the experiment, we fine-tune the ViT for two epochs. After fine-tuning, the models achieved test accuracies of 91.18% on C100 and 86.34% on TinyImg. These models serve as the original models to be repaired. Our setup reflects more realistic scenarios in which even well-trained models may still need to be repaired for specific types of misbehavior.

D. Fault Benchmarks

A fault benchmark refers to a set of misclassified samples that are targeted for repair. The target misclassification is identified by specifying its *type* and *rank*.

First, we define three types of misclassification based on the pair of predicted and true labels $(c_{\text{pred}}, c_{\text{true}})$ as follows:

1) *SRC-TGT.* Misclassification where the model predicts a specific class src for inputs whose true label is another specific class tgt:
$$(c_{\text{pred}}, c_{\text{true}}) = (src, tgt), \quad \text{where } src \neq tgt.$$
2) *TGT-FP* (False positives for tgt). Misclassification where the model wrongly predicts tgt for inputs not in tgt:
$$\exists y \neq tgt, \quad (c_{\text{pred}}, c_{\text{true}}) = (tgt, y).$$
3) *TGT-FN* (False negatives for tgt). Misclassification where the model fails to predict tgt for inputs labeled as tgt:
$$\exists y' \neq tgt, \quad (c_{\text{pred}}, c_{\text{true}}) = (y', tgt).$$

We consider these benchmark types to reflect real-world scenarios where developers are particularly concerned about classes that produce a lot of false positives or false negatives.

[2]https://huggingface.co/google/vit-base-patch16-224

Next, we define the rank of misclassification for each type of misclassification. For the *SRC-TGT*, we count the number of misclassified samples for each pair of (*src*, *tgt*) and rank these pairs in descending order of frequency. This allows us to identify the most frequent misclassification patterns between specific class pairs. For the *TGT-FP* and *TGT-FN*, we compute the precision and recall for each class, respectively, and rank the classes in ascending order. In our experiments, we target the top-3 ranks (i.e., rank 1, 2, and 3). By specifying the misclassification type and its rank, we can identify a small set of target samples that represent the target misclassification. This set corresponds to $\mathcal{I}_{\text{mis}}$ in Sect. III-A1, while the set of correctly classified samples corresponds to $\mathcal{I}_{\text{cor}}$.

Tab. I shows the number of misclassified samples used in this study. We obtain a total of 18 fault benchmarks (2 datasets $\times$ 3 types $\times$ 3 ranks). While the misclassification types and ranks are identified based on the repair set, the generalization performance of the repair is evaluated on a disjoint test set. Therefore, for each type and rank of misclassification in the repair set (i.e., $|\mathcal{I}_{\text{mis}}^{\text{repair}}|$), we also report the number of corresponding misclassified samples in the test set (i.e., $|\mathcal{I}_{\text{mis}}^{\text{test}}|$).

E. Baselines

The first baseline is a *random* strategy, where a fixed number of weights are randomly chosen from the set $W_{\text{bef}} \cup W_{\text{aft}}$.

The second baseline is ARACHNE [5], a state-of-the-art DNN repair method that selects weights on the Pareto front defined by the forward impact and the gradient loss (gray blocks in Fig. 2). Although the original ARACHNE does not support Transformer models, we apply it to the FFN layers, which can be regarded as feed-forward networks composed of two linear transformations (i.e., W_{bef} and W_{aft}).

The third baseline is PRoViT [10], a repair method designed specifically for ViTs, which we use in RQ5. We evaluate its three variants (PRoViT$_{\text{LP}}$, PRoViT$_{\text{FT}}$, and PRoViT$_{\text{FT+LP}}$), all targeting the same final-block FFN as REPTRAN so that the methods differ only in how they modify that module.

F. Configuration of Repair Methods

Selection Phase. The parameter p in Eq. 2, which balances the contributions of ModFI and ModGL in the suspiciousness score, is set to 0.5 for RQ1–3. This means that the ModFI and ModGL are considered equally important. We investigate the sensitivity of this choice in RQ4 ($p \in \{0.1, 0.5, 0.9\}$).

The number of selected weights N_w is set to one of the values in {11, 236, 472, 944}. Here, $N_w = 11$ corresponds to the average number of weights selected by ARACHNE. In contrast, $N_w = 236, 472, 944$ correspond to 0.005%, 0.01%, and 0.02% of the total number of weight parameters in a FFN layer, respectively, and are used in RQ3 to analyze the impact of the number of selected weights.

We target only the final layer of the Transformer encoder block (i.e., $L = 12$) for repair. Although our method can, in principle, be extended to other layers or even multiple layers, we adopt this design for the following reasons. First, we follow prior work such as PRoViT [10], which also focuses on the

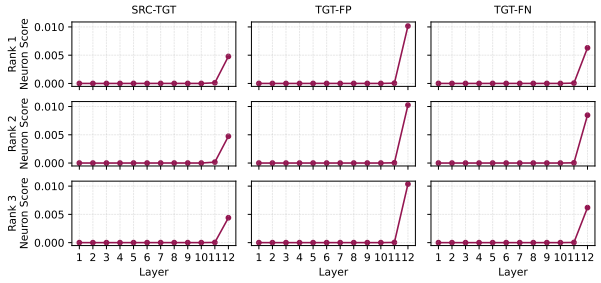


Fig. 3. Layer-wise mean neuron scores for C100.

final layer to reduce the number of target parameters. Second, a recent study [20] revealed that most of the knowledge neurons are concentrated in the final layer, suggesting the importance of this layer. Therefore, our choice is not only practically motivated but also supported by prior findings that highlight the semantic significance of the final layer.

Moreover, to empirically support this design choice, we conducted an experiment to examine which layers are most suitable for repair. We computed the neuron scores across all layers. Fig. 3 presents the layer-wise mean neuron scores for each misclassification type and rank on C100; TinyImg exhibited the same trend and is omitted for space, with the full results available in our replication package. We observe that the final layer consistently shows significantly higher scores than the preceding layers across misclassification types and ranks. This observation is similar to prior findings [20] that emphasize the concentration of the knowledge neurons in the final layer. This suggests that the final layer captures the most distinguishable behavior between correctly and incorrectly classified inputs, making it a prime target for repair.

Search Phase. We use the same DE parameters as ARACHNE [5], including the population size N_p, scaling factor F, and crossover rate CR. The parameters $G_{\max}$ and G_{stag}, which control the termination conditions of the search phase, are empirically set to 50 and 10, respectively, to avoid excessive repair time. The coefficient α in the fitness function (Eq. 3) balances the trade-off between maintaining correct behavior and reducing misbehavior. We investigate the impact of α on the performance of the patched model in RQ4. In the other RQs, we fix it to 10, following prior studies [5], [34]. To account for the randomness in the search phase introduced by DE, we repeat each experiment five times (#runs=5) under the same configuration.

G. Evaluation Metrics

We apply each repair method on the repair set and evaluate the patched model M' on the test set, using three metrics: *repair rate* (RR), *break rate* (BR), and *total execution time* (T_{tot}). Let $\mathcal{I}^{\text{test}}_{\text{cor}}$ and $\mathcal{I}^{\text{test}}_{\text{mis}}$ denote the test-set samples correctly classified by the original model and the target misclassified samples, respectively.

The repair rate (RR) is the fraction of target misclassifications corrected in the test set:

$$RR = \frac{|\{(x,y) \in \mathcal{I}^{\text{test}}_{\text{mis}} \mid M'(x) = y\}|}{|\mathcal{I}^{\text{test}}_{\text{mis}}|}.$$

TABLE II

RQ1 - REPAIR GENERALIZATION PERFORMANCE. ROW-WISE RANKING IS HIGHLIGHTED WITH GOLD (1ST), SILVER (2ND), AND BRONZE (3RD). ARROWS ↑ (↓) INDICATE THAT HIGHER (LOWER) VALUES ARE BETTER.

Dataset	Rank / Type	REPTRAN		ARACHNE		Random$_R$		Random$_A$	
		$RR\uparrow$	$BR\downarrow$	$RR\uparrow$	$BR\downarrow$	$RR\uparrow$	$BR\downarrow$	$RR\uparrow$	$BR\downarrow$
C100	1 / *SRC-TGT*	95.2%	0.9%	19.0%	0.2%	6.7%	0.2%	0.0%	0.0%
	1 / *TGT-FP*	68.2%	1.4%	17.6%	0.2%	7.6%	0.2%	0.0%	0.0%
	1 / *TGT-FN*	94.3%	1.3%	25.0%	0.2%	7.9%	0.2%	0.0%	0.0%
	2 / *SRC-TGT*	80.0%	0.9%	37.5%	0.1%	12.5%	0.1%	7.5%	0.0%
	2 / *TGT-FP*	52.2%	1.4%	13.0%	0.1%	1.7%	0.1%	0.0%	0.0%
	2 / *TGT-FN*	92.8%	2.4%	20.0%	0.2%	16.8%	0.2%	0.0%	0.0%
	3 / *SRC-TGT*	82.5%	0.9%	0.0%	0.1%	0.0%	0.1%	0.0%	0.0%
	3 / *TGT-FP*	68.6%	1.3%	7.1%	0.0%	4.3%	0.1%	0.0%	0.0%
	3 / *TGT-FN*	86.9%	1.6%	6.9%	0.1%	6.9%	0.1%	0.0%	0.0%
TinyImg	1 / *SRC-TGT*	91.4%	0.7%	28.6%	0.1%	21.4%	0.2%	0.0%	0.0%
	1 / *TGT-FP*	14.4%	1.2%	9.6%	0.3%	4.0%	0.2%	0.0%	0.0%
	1 / *TGT-FN*	78.0%	1.4%	20.0%	0.1%	16.0%	0.2%	1.0%	0.0%
	2 / *SRC-TGT*	90.0%	0.7%	33.3%	0.1%	15.0%	0.2%	0.0%	0.0%
	2 / *TGT-FP*	42.7%	1.1%	0.0%	0.1%	0.0%	0.2%	0.0%	0.0%
	2 / *TGT-FN*	90.8%	5.7%	16.9%	0.4%	0.0%	0.2%	0.0%	0.0%
	3 / *SRC-TGT*	86.7%	0.4%	44.4%	0.1%	20.0%	0.2%	0.0%	0.0%
	3 / *TGT-FP*	39.2%	0.8%	2.4%	0.2%	5.6%	0.2%	0.0%	0.0%
	3 / *TGT-FN*	90.0%	3.1%	6.2%	0.1%	3.8%	0.2%	0.0%	0.0%
Average		74.7%	1.5%	17.1%	0.2%	8.3%	0.2%	0.5%	0.0%

The break rate (BR) captures the side effects of the repair, i.e., the fraction of originally correct samples that the patched model misclassifies:

$$BR = \frac{|\{(x,y) \in \mathcal{I}^{\text{test}}_{\text{cor}} \mid M'(x) \neq y\}|}{|\mathcal{I}^{\text{test}}_{\text{cor}}|}.$$

The total repair time (T_{tot}) is the overall time for the repair process. For ARACHNE and REPTRAN, it is the sum of the selection phase (T_{select}) and the search phase (T_{repair}); for the random baseline, T_{tot} equals the search time.

V. EVALUATION

All experiments were run on a machine equipped with an Intel Core i9-14900K CPU and an NVIDIA GeForce RTX 4090 GPU. Detailed version information for all libraries is available in our replication package.

A. RQ1: Effectiveness of REPTRAN

Approach. We evaluate REPTRAN by comparing it with ARACHNE and random baselines. REPTRAN selects 472 weights (0.01% of FFN parameters), whereas ARACHNE selects 11 on average. Since ARACHNE selects weights on the Pareto front, the number of selected weights cannot be controlled. Experiments that control the number of selected weights are conducted in RQ3; in RQ1, we configure each method to reflect settings under which it is expected to perform well. We also prepare two random baselines: Random$_R$, which selects the same number of weights as REPTRAN (472), and Random$_A$, which selects the same number as ARACHNE (11), to assess whether each method outperforms random selection under an equivalent weight budget.

Results. Tab. II shows the repair generalization performance of each method on C100 and TinyImg. Each cell in the table reports the average value over five runs.

REPTRAN achieves the highest repair rate in all 18 cases. Its performance margin over the second-best method is substantial in most cases. At the same time, REPTRAN

consistently records the largest break rate. This suggests that it tends to select weights that strongly influence model outputs. Such weights are effective in correcting misclassifications, but they also increase the risk of breaking correct classifications. Even so, the trade-off appears reasonable, given that the maximum break rate stays below 5.7%, while the maximum repair rate reaches as high as 95.2%.

ARACHNE exhibits a more conservative behavior than REPTRAN. ARACHNE has a performance level between REPTRAN and the random baselines. While its repair rate is substantially lower than that of REPTRAN, it still outperforms the random baselines. In fact, ARACHNE ranks second in repair rate in 17 out of 18 cases. This result suggests that selecting weights based on the Pareto front may be too limited to achieve substantial repairs for the FFNs in Transformer models, though it is more effective than random selection.

Both REPTRAN and ARACHNE exhibit higher repair and break rates compared to the random baselines, as seen in the comparisons between REPTRAN vs. Random$_R$ and ARACHNE vs. Random$_A$. This suggests that they can identify weights that are more likely to contribute to fixing misclassifications than those selected randomly, demonstrating the effectiveness of the selection phase. At the same time, the result indicates the inherent difficulty of avoiding side effects when selecting such influential weights.

To statistically evaluate the differences in repair and break rates across repair methods, we performed the Wilcoxon signed-rank test [35]. As shown in Tab. III, we compared each pair of methods for every dataset, using 9 metric values per comparison (3 ranks $\times$ 3 misclassification types, where each value represents the average across 5 runs). The table reports the statistical significance and the effect size, measured by Cliff's δ [36]. It ranges from -1 to $+1$: positive values indicate that the first method in Tab. III tends to have larger metric values than the second, whereas negative values indicate the opposite. An effect size is considered large when $|\delta| > 0.474$ [37]. All reported p-values are corrected using the Holm method [38] to control the family-wise error rate.

Tab. III shows that high repair and break rates of REPTRAN are statistically significant. In fact, all comparisons involving REPTRAN yield an effect size of 1.00 and $p < .05$. This trade-off between repair and break rates is also observed in other comparisons. The only exception is found in the comparison between ARACHNE and Random$_R$, where ARACHNE shows a higher repair rate and a lower break rate based on the sign of the effect size; however, the effect sizes are smaller than those involving REPTRAN, and the differences in break rate are not statistically significant.

Answer to RQ1: REPTRAN outperforms all baselines in repair rate across all 18 cases, achieving up to 95.2%. Its superior effectiveness comes with a higher break rate, though the maximum remains below 5.7%. Statistical tests confirm that these differences are statistically significant in all comparisons involving REPTRAN.

TABLE III
RQ1 - WILCOXON SIGNED-RANK TESTS ON RR AND BR. EACH CELL REPORTS CLIFF'S δ. SIGNIFICANCE LEVELS: $^{*}p<.05$, $^{**}p<.01$. ABBREVIATIONS: REP = REPTRAN, ARA = ARACHNE, RAND$_R$ = RANDOM$_R$, RAND$_A$ = RANDOM$_A$.

Dataset	Metric	Rep vs. Ara	Rep vs. Rand$_R$	Rep vs. Rand$_A$	Ara vs. Rand$_R$	Ara vs. Rand$_A$	Rand$_R$ vs. Rand$_A$
C100	$RR\uparrow$	+1.00 *	+1.00 *	+1.00 *	+0.78 *	+0.89 *	+0.89 *
	$BR\downarrow$	+1.00 *	+1.00 *	+1.00 *	-0.33	+1.00 *	+1.00 *
TinyImg	$RR\uparrow$	+1.00 *	+1.00 *	+1.00 *	+0.67 *	+0.89 *	+0.78 *
	$BR\downarrow$	+1.00 *	+1.00 *	+1.00 *	-0.56	+1.00 *	+1.00 *

TABLE IV
RQ2 - EXECUTION TIME FOR EACH METHOD ON C100 AND TINYIMG. ROW-WISE RANKING OF T_{TOT} IS HIGHLIGHTED WITH GOLD (1ST), SILVER (2ND), AND BRONZE (3RD). ABBREVIATIONS: REP = REPTRAN, ARA = ARACHNE, RAND$_R$ = RANDOM$_R$, RAND$_A$ = RANDOM$_A$.

Dataset	Rank / Type	T_{select} [sec.]		T_{repair} [sec.]		T_{tot} [sec.]			
		Rep	Ara	Rep	Ara	Rep	Ara	Rand$_R$	Rand$_A$
C100	1 / *SRC-TGT*	21.06	**18.55**	**329.99**	405.18	351.04	423.72	418.875	241.674
	1 / *TGT-FP*	**11.73**	13.86	**386.89**	407.92	398.61	421.78	423.179	221.351
	1 / *TGT-FN*	**11.66**	17.90	**281.87**	408.97	293.53	426.87	414.403	341.957
	2 / *SRC-TGT*	**11.68**	16.94	**264.59**	404.16	276.27	421.09	417.089	293.511
	2 / *TGT-FP*	**11.78**	15.55	**331.96**	417.26	343.74	432.81	386.160	273.072
	2 / *TGT-FN*	**11.68**	17.06	**287.50**	418.63	299.18	435.69	413.794	307.170
	3 / *SRC-TGT*	**11.76**	15.83	**296.74**	404.27	308.50	420.10	409.712	280.190
	3 / *TGT-FP*	**11.73**	17.32	**332.61**	414.86	344.34	432.18	386.064	237.540
	3 / *TGT-FN*	**11.74**	17.17	**370.24**	403.76	381.98	420.93	421.981	283.828
TinyImg	1 / *SRC-TGT*	**32.73**	35.00	**511.41**	808.01	544.14	843.01	820.283	633.284
	1 / *TGT-FP*	**21.60**	24.25	**646.00**	841.71	667.60	865.96	858.033	473.249
	1 / *TGT-FN*	**21.14**	23.02	**539.31**	816.44	560.45	839.47	780.018	621.619
	2 / *SRC-TGT*	**21.01**	27.55	**626.97**	708.07	647.98	735.61	824.027	528.644
	2 / *TGT-FP*	**21.30**	22.79	**597.41**	826.76	618.70	849.56	773.139	655.785
	2 / *TGT-FN*	**21.18**	24.27	**728.61**	744.41	749.79	768.67	773.695	618.982
	3 / *SRC-TGT*	**21.08**	23.71	**347.78**	808.91	368.85	832.62	821.061	507.497
	3 / *TGT-FP*	**21.17**	23.31	**640.50**	790.13	661.67	813.44	806.136	711.400
	3 / *TGT-FN*	**21.09**	23.47	**733.97**	769.84	755.06	793.31	820.926	519.099
Average		17.62	20.97	458.57	599.96	476.19	620.93	609.37	430.547

B. RQ2: Efficiency of REPTRAN

Approach. We use the same patched models as in RQ1 but focus on the total repair time instead of repair and break rates. For REPTRAN and ARACHNE, we report the total repair time as well as the time spent in the selection and search phases. For the random baselines, only the total repair time is reported.

Results. Tab. IV shows the total repair time of REPTRAN, ARACHNE, Random$_R$, and Random$_A$ on C100 and TinyImg. Each cell in the table reports the average value over five runs. Bold numbers in the T_{select} and T_{repair} columns mark the faster method between REPTRAN and ARACHNE for each case. We also show the statistical significance and the effect size of the differences in the total repair time across methods in Tab. V.

From Tab. IV, REPTRAN is consistently faster than ARACHNE across all 18 cases. When examined by phase, REPTRAN is faster than ARACHNE in the search phase in all cases, and also in the selection phase in 17 out of 18 cases. Wilcoxon test results in Tab. V confirm this advantage: for both datasets, the RepTran vs. ARACHNE comparison yields an effect size of -1.00 with statistical significance ($p<.05$).

This efficiency is notable given that the number of selected weights, which is the number of variables in the DE algorithm, is larger for REPTRAN than for ARACHNE (472 vs. 11). This is not a trivial outcome, as shown by the comparison between Random$_R$ and Random$_A$ in the rightmost column of Tab. V, where Random$_R$, which uses more variables, is significantly slower. These results suggest that REPTRAN effectively iden-

TABLE V

RQ2 - WILCOXON SIGNED-RANK TESTS ON T_{TOT}. EACH CELL REPORTS CLIFF'S δ TOGETHER WITH STATISTICAL SIGNIFICANCE. SIGNIFICANCE LEVELS: $^{*}p<.05$, $^{**}p<.01$. ABBREVIATIONS: REP = REPTRAN, ARA = ARACHNE, RAND_R = RANDOM_R, RAND_A = RANDOM_A.

Dataset	Rep vs. Ara	Rep vs. Rand_R	Rep vs. Rand_A	Ara vs. Rand_R	Ara vs. Rand_A	Rand_R vs. Rand_A
C100	-1.00 *	-1.00 *	$+0.33$	-0.33	$+1.00$ *	$+1.00$ *
TinyImg	-1.00 *	-1.00 *	-0.11	-0.56	$+1.00$ *	$+1.00$ *

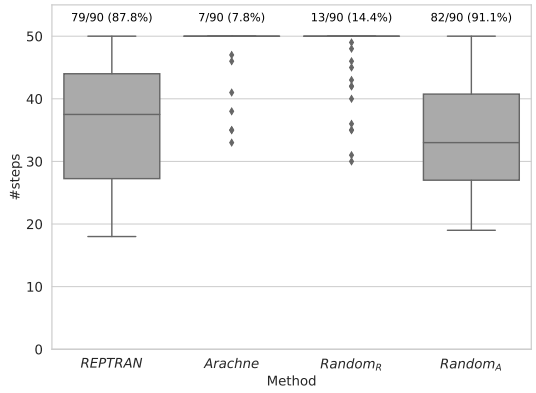


Fig. 4. RQ2 - Number of steps taken in the search phase for each method ($G_{\max} = 50$, $G_{\text{stag}} = 10$). Each box represents the distribution over 90 data points (2 datasets $\times$ 3 misclassification types $\times$ 3 ranks $\times$ 5 runs). The numbers above the boxes indicate the frequency of early stopping.

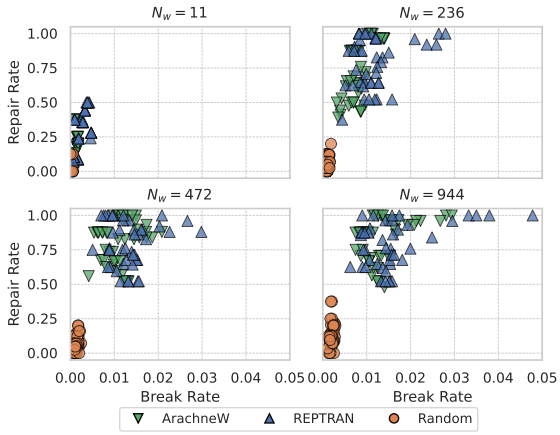
tifies weights for faster convergence, enabling early stopping and shorter repair time despite handling more variables.

To further examine the efficiency of each method, we analyze the number of steps for the search phase, as shown in Fig. 4. **REPTRAN terminated early in 87.8% of runs, accounting for its short repair time.** Similarly, Random_A, which also exhibited short repair time, triggered early stopping in 91.1% of the runs. However, the reasons for early stopping differ between REPTRAN and Random_A, especially in light of the results from RQ1. For REPTRAN, early termination is due to rapid convergence to high fitness values, which aligns with its strong repair performance observed in RQ1. By contrast, Random_A stops early because the selected weights have minimal effect on models, resulting in little to no improvement in fitness and thus early termination for a negative reason.

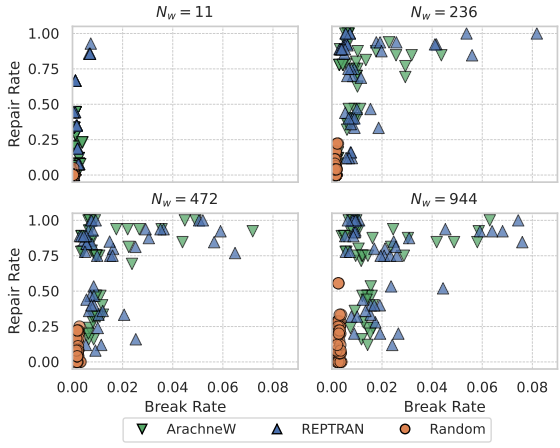
Answer to RQ2: REPTRAN is faster than ARACHNE in all 18 cases, despite optimizing more variables (472 vs. 11). It achieves early stopping in 87.8% of runs, enabling faster convergence and efficient repair.

C. RQ3: Effect of the Number of Selected Weights

Approach. The number of selected weights can significantly impact the outcome. Selecting too few weights may lead to insufficient repair, whereas selecting too many may introduce unintended side effects. To investigate this trade-off, we conducted experiments by varying the number of selected weights: 11 (the average number selected by ARACHNE), 236, 472, and 944 (corresponding to 0.005%, 0.01%, and 0.02% of all weights, respectively), as described in Sect. IV-F. To enable control over the number of selected weights in ARACHNE, we implement a variant called ARACHNEW, which selects



(a) C100



(b) TinyImg

Fig. 5. RQ3 - RR and BR for different methods and number of weights.

the top-N_w weights with the highest average score of forward impact and gradient loss, instead of relying on the Pareto front.

Results. Fig. 5 shows the repair and break rates for each number of selected weights. To directly evaluate the trade-off between repair and break rates, we compute the harmonic mean of RR and $1 - BR$.[3] The resulting scores are summarized in Tab. VI.

Improvements in repair rate are often accompanied by an increase in break rate. From Fig. 5, both REPTRAN and ARACHNEW show higher repair rate when more than 236 weights are modified, but this typically results in a rise in break rate as well. While some configurations (especially on TinyImg) achieve favorable combinations of high repair rate and low break rate, the overall tendency reveals a trade-off between maximizing repair effectiveness and minimizing unintended side effects.

The trade-off between the repair and break rates is most effectively balanced at $N_w = 944$, but the improvement saturates after $N_w = 236$, indicating a diminishing benefit from selecting more weights. As shown in the rightmost column of Tab. VI, the average harmonic mean reaches its

[3]Since lower break rates are better, we use the complement ($1 -$ break rate) so that higher values indicate better performance, similar to the repair rate.

TABLE VI
RQ3 - HARMONIC MEAN OF RR AND $1 - BR$.

Dataset	N_w	REPTRAN	ARACHNEW	Random	Average
C100	11	0.494	0.263	0.017	0.258
	236	0.910	0.853	0.097	0.620
	472	0.914	0.908	0.141	0.655
	944	0.909	0.921	0.254	0.695
TinyImg	11	0.430	0.313	0.005	0.249
	236	0.810	0.789	0.118	0.572
	472	0.814	0.811	0.189	0.605
	944	0.819	0.822	0.276	0.639

TABLE VII
RQ3 - WILCOXON SIGNED-RANK TESTS ON RR AND BR. EACH CELL REPORTS CLIFF'S δ TOGETHER WITH STATISTICAL SIGNIFICANCE. SIGNIFICANCE LEVELS: $^{*}p<.05$, $^{**}p<.01$. ABBREVIATIONS: REP = REPTRAN, ARAW = ARACHNEW, RAND = RANDOM.

Dataset	N_w	RR			BR		
		Rep vs. AraW	Rep vs. Rand	AraW vs. Rand	Rep vs. AraW	Rep vs. Rand	AraW vs. Rand
C100	11	+0.67*	+1.00*	+0.89*	+1.00*	+1.00*	+1.00*
	236	+0.56	+1.00*	+1.00*	+0.33	+1.00*	+1.00*
	472	+0.11	+1.00*	+1.00*	+0.56	+1.00*	+1.00*
	944	−0.44	+1.00*	+1.00*	+0.33	+1.00*	+1.00*
TinyImg	11	+0.33	+0.89*	+0.89*	−0.11	+1.00*	+1.00*
	236	+0.22	+1.00*	+1.00*	+0.56	+1.00*	+1.00*
	472	+0.22	+1.00*	+1.00*	+0.33	+1.00*	+1.00*
	944	+0.33	+1.00*	+1.00*	+0.78*	+1.00*	+1.00*

maximum at $N_w = 944$, suggesting that this setting achieves the best balance between improving the repair rate and minimizing the break rate. However, increasing N_w from 236 to 472 or 944 results in only marginal gains in the harmonic mean. This suggests that selecting more than 236 weights provides limited additional benefit in optimizing the trade-off.

Tab. VII shows the results of our statistical comparison of repair and break rates, as in the previous RQs. Statistically significant differences ($p < .05$) accompanied by a large positive effect size ($\delta > 0.474$) are highlighted in yellow. For all weight settings, REPTRAN and ARACHNEW consistently produced statistically significantly higher repair and break rates than the random baseline, with all comparisons showing large effect sizes and $p < .05$.

REPTRAN statistically significantly outperforms ARACHNEW in only one out of eight comparisons for the repair rate (C100 with 11 weights). However, seven out of eight cases show a positive effect size, with large effects observed when the number of selected weights is small (e.g., for C100, $\delta = +0.67$ with 11 weights and $\delta = +0.56$ with 236 weights). This indicates a general tendency for REPTRAN to yield higher repair rates than ARACHNEW, though the statistical support is limited.

Answer to RQ3: While 944 weights achieved the best trade-off between repair and break rates, improvements largely saturated at 236, suggesting limited benefit from adding more. REPTRAN generally outperformed ARACHNEW in the repair rate, showing positive effect sizes in seven out of eight comparisons.

D. RQ4: Effect of the Balance Coefficients

Approach. REPTRAN uses one balance coefficient in each phase. The coefficient p in the selection phase (Eq. 2) balances ModFI and ModGL when scoring weights, whereas α in the search phase (Eq. 3) balances maintaining correct behavior and reducing misbehavior. In this RQ, we analyze the effect of each coefficient in isolation by varying p over $\{0.1, 0.5, 0.9\}$ and α over $\{1, 2, 4, 8, 10\}$, following the same setup as in the original ARACHNE experiments for α [5]. We fix the number of weights at $N_w = 236$, which showed a good balance in RQ3. This allows us to eliminate the influence of the number of selected weights and measure the effect of each balance coefficient.

TABLE VIII
RQ4 - KRUSKAL–WALLIS TEST RESULTS FOR RR AND BR ACROSS DIFFERENT BALANCE-COEFFICIENT VALUES.

Dataset	p for selection phase		α for search phase	
	RR p-value	BR p-value	RR p-value	BR p-value
C100	0.404	0.614	0.996	0.984
TinyImg	0.997	0.468	0.994	0.939

Results. For space, we do not report the raw repair and break rates for each value of p and α here; these are available in our replication package, and we focus on the results of the statistical tests. We conducted a Kruskal–Wallis test [39] to determine whether the distributions of repair or break rates differ across the tested values of each balance coefficient. The null hypothesis for each dataset, metric, and coefficient states that all groups for the coefficient come from the same distribution, i.e., the repair or break rates are identically distributed regardless of the coefficient value.

The selection-phase coefficient p did not significantly affect repair or break rates. Across the tested values of p, no clear trend emerged in either repair or break rates. As shown in Tab. VIII, all null hypotheses were retained for both datasets and both metrics. This result indicates that the choice of p has little impact on the repair outcome.

No statistically significant differences were observed in the repair or break rates across different values of α. As shown in Tab. VIII, all null hypotheses were retained for α, with p-values approaching 1.0 in every case. This indicates not only a lack of statistically significant differences across α values, but also that the observed repair and break rates are highly consistent with the assumption of identical distributions. These findings further support the conclusion that the balance coefficient α does not have a meaningful impact on repair or break rates.

Answer to RQ4: Varying the balance coefficients p from 0.1 to 0.9 and α from 1 to 10 had minimal impact on repair and break rates. Kruskal–Wallis tests showed no statistically significant differences for either coefficient, indicating that p and α do not meaningfully affect the outcome.

E. RQ5: Comparison with a ViT-specific Method

Approach. We compare REPTRAN with the three PRoViT variants, all targeting the same final-block FFN. We use the same 18 benchmarks as in previous RQs and report each metric as the percentage change $(\text{PRoViT}/\text{REPTRAN} - 1) \times 100$, where a positive change means PRoViT repairs more (better), breaks more (worse), or is slower (worse), respectively. We set a time budget of 1,800 seconds for solving the LP and treat any

TABLE IX

RQ5 - REPAIR RATE (RR), BREAK RATE (BR), AND REPAIR TIME (T_{TOT}) OF THE THREE PROVIT VARIANTS RELATIVE TO REPTRAN. ARROWS ↑ (↓) INDICATE THAT HIGHER (LOWER) VALUES ARE BETTER. TO DENOTES A TIME OUT.

Dataset / Rank / Type	PRoViT$_{\text{LP}}$			PRoViT$_{\text{FT}}$			PRoViT$_{\text{FT+LP}}$		
	$RR\uparrow$	$BR\downarrow$	$T_{\text{tot}}\downarrow$	$RR\uparrow$	$BR\downarrow$	$T_{\text{tot}}\downarrow$	$RR\uparrow$	$BR\downarrow$	$T_{\text{tot}}\downarrow$
C100 / 1 / *SRC-TGT*	+5%	+28%	−48%	+5%	+28%	−100%	+5%	+16%	−46%
C100 / 1 / *TGT-FP*	−18%	+9%	+279%	−18%	−18%	−99%	−22%	−23%	+55%
C100 / 1 / *TGT-FN*	+6%	+77%	+23%	+6%	+145%	−100%	+6%	+87%	+8%
C100 / 2 / *SRC-TGT*	+9%	−20%	−8%	+25%	+18%	−100%	+9%	−37%	+16%
C100 / 2 / *TGT-FP*	+8%	−35%	+163%	+8%	−22%	−99%	−2%	−37%	+136%
C100 / 2 / *TGT-FN*	+8%	−17%	+221%	+8%	+43%	−100%	+8%	−14%	+75%
C100 / 3 / *SRC-TGT*	+21%	−16%	−47%	+21%	+8%	−100%	+21%	−10%	−24%
C100 / 3 / *TGT-FP*	−13%	+54%	+98%	+8%	−5%	−97%	−17%	+9%	+113%
C100 / 3 / *TGT-FN*	−1%	−29%	+102%	+11%	+34%	−100%	+1%	−25%	+79%
TinyImg / 1 / *SRC-TGT*	TO			+9%	−4%	−100%	+9%	−4%	−100%
TinyImg / 1 / *TGT-FP*	TO			+67%	−11%	−99%	+50%	−59%	+182%
TinyImg / 1 / *TGT-FN*	TO			−1%	−41%	−100%	−4%	−47%	−68%
TinyImg / 2 / *SRC-TGT*	+11%	+26%	+18%	+11%	−8%	−100%	+11%	−15%	+28%
TinyImg / 2 / *TGT-FP*	TO			+9%	−8%	−100%	+19%	−67%	+203%
TinyImg / 2 / *TGT-FN*	TO			+10%	+27%	−100%	−3%	−54%	+48%
TinyImg / 3 / *SRC-TGT*	+3%	−39%	+57%	+3%	−20%	−100%	+3%	−20%	−100%
TinyImg / 3 / *TGT-FP*	−39%	+85%	+173%	+0%	+218%	+172%	−4%	−1%	+160%
TinyImg / 3 / *TGT-FN*	+11%	+166%	+137%	+4%	−15%	−100%	−3%	−67%	+146%
Average	**+1%**	**+22%**	**+90%**	**+10%**	**+20%**	**−84%**	**+5%**	**−20%**	**+51%**

run exceeding it as a timeout (TO). This budget is generous, as it is more than twice the largest repair time observed for any method in RQ2, which stays below 1,000 seconds (Tab. IV).

Results. Tab. IX reports how each PRoViT variant compares with REPTRAN on repair rate (RR), break rate (BR), and repair time (T_{tot}). A green (red) cell indicates that the variant is better (worse) than REPTRAN on that metric, and the bottom row aggregates these changes over the benchmarks.

On average, no PRoViT variant improves over REPTRAN on all three metrics at once. As the Average row shows, each variant is worse than REPTRAN on at least one metric: PRoViT$_{\text{LP}}$ on both break rate (+22%) and time (+90%), PRoViT$_{\text{FT}}$ on break rate (+20%), and PRoViT$_{\text{FT+LP}}$ on time (+51%). That is, the gains in repair rate that the variants achieve come at the cost of either more broken behaviors or substantially longer repair time.

PRoViT$_{\text{LP}}$ and PRoViT$_{\text{FT+LP}}$ are slower than REPTRAN and do not scale to larger label spaces. These two variants are on average +90% and +51% slower, as both are bottlenecked by solving the LP. This cost grows with the label space. On TinyImg the slowdown reaches up to +203%, and PRoViT$_{\text{LP}}$ fails to return a solution in 5 of the 9 TinyImg benchmarks. REPTRAN has no such scalability wall, since it performs a localized repair rather than solving a global LP.

PRoViT$_{\text{FT}}$ is the only variant faster than REPTRAN, but at the cost of overfitting. It fine-tunes until the few dozen samples in the repair set (at most 56, as shown in Tab. I) are fully fixed, which explains both its speed (−84%) and its higher break rate (+20%). Fixing the repair set does not by itself guarantee generalization to unseen inputs.

Answer to RQ5: No PRoViT variant improves over REPTRAN on all of repair rate, break rate, and time on average: each is worse on at least one. REPTRAN attains comparable repair and break rates while running 51 to 90% faster than the LP-based variants, one of which (PRoViT$_{\text{LP}}$) times out on 5 of the 9 TinyImg benchmarks; the only faster variant, PRoViT$_{\text{FT}}$, achieves its speed by overfitting the repair set.

TABLE X

COMPONENT ABLATION OF THE NEURON SCORE. SIGNIFICANCE LEVELS: $^{*}p<.05$, $^{**}p<.01$, $^{***}p<.001$.

Dataset	Contrast	RR	BR
C100	Full vs. VDiff-only	+0.18	−0.22
	Full vs. MisAct-only	−0.18	−0.11
TinyImg	Full vs. VDiff-only	$+0.42^{***}$	$+0.49^{**}$
	Full vs. MisAct-only	+0.07	+0.13

VI. DISCUSSION

A. Contribution of the Neuron Score Components

As described in Sect. III-A1, the neuron score is the product of two components, *VDiff* and *MisAct*. To justify our proposed neuron score, we isolate the contribution of each component by ablating it at a fixed budget of $N_w = 236$, which showed a good balance between repairs and breaks in RQ3. Keeping all else equal, we compare *Full* (VDiff × MisAct), *VDiff-only*, and *MisAct-only* over the 18 fault benchmarks. For each dataset, we statistically compare the RR and BR of Full against those of each ablated variant (VDiff-only and MisAct-only) using paired Wilcoxon signed-rank tests over the 45 paired values (9 benchmarks × 5 runs, Holm-corrected). Tab. X reports the resulting Cliff's δ, where a positive δ means Full attains a higher RR (or BR) than the ablated variant.

MisAct is the dominant component, while VDiff is complementary. Removing MisAct (VDiff-only) significantly lowers RR on TinyImg ($\delta = 0.42$, $p < .001$) and also on C100 ($\delta = 0.18$, not significant). Moreover, as the signs of δ in Tab. X show, no ablated variant attains both a higher RR and a lower BR than Full; any gain on one metric is offset by a loss on the other. Combining the two components therefore yields a reasonably balanced selection across both datasets.

B. How REPTRAN Differs from ARACHNEW and PRoViT?

Although REPTRAN and ARACHNEW showed no statistically significant differences in most RQ3 settings, and RQ5 showed that no PRoViT variant dominates REPTRAN across all metrics, understanding how these methods differ is important for assessing their complementarity.

1) Differences in the selected weights: We computed the Jaccard coefficients between the weights selected by REPTRAN and ARACHNEW across various configurations. The Jaccard coefficient quantifies the degree of overlap between two sets, ranging from 0 (no overlap) to 1 (complete overlap).

The median Jaccard coefficient across all configurations was 0.13, ranging from 0.00 to 0.47, indicating that the two methods often selected different weights. The highest coefficients were 0.47 for TinyImg, Rank 1, and *TGT-FP*, suggesting that the methods may converge on similar weights for some misbehaviors. Overall, however, the low overlap supports that incorporating neuron scores in REPTRAN yields weight selections qualitatively different from ARACHNEW.

2) Differences in repaired and broken samples: To examine whether the methods repair and break different samples, we compared the sets of test samples that REPTRAN, ARACHNEW, and PRoViT$_{\text{FT+LP}}$ consistently repaired or broke across

TABLE XI
NUMBER OF UNIQUE REPAIRS AND BREAKS. EACH CELL LISTS COUNTS IN THE ORDER REPTRAN/ARACHNEW/PROVIT$_{\mathrm{FT+LP}}$; BOLD VALUES INDICATE THE HIGHEST COUNT IN THAT CELL.

Dataset / Type	Unique Repairs	Unique Breaks
C100 / *SRC-TGT*	0/0/**1**	4/3/**49**
C100 / *TGT-FN*	0/0/**8**	19/7/**199**
C100 / *TGT-FP*	4/4/**12**	4/11/**105**
TinyImg / *SRC-TGT*	0/0/**2**	3/2/**48**
TinyImg / *TGT-FN*	1/0/**3**	**101**/22/57
TinyImg / *TGT-FP*	0/1/**5**	6/36/**67**

all five runs. Among the three PRoViT variants, we focus on PRoViT$_{\mathrm{FT+LP}}$ because it is the best-performing one in RQ5. We focus on such consistently repaired or broken samples to capture the inputs that each method reliably repairs or breaks rather than incidental cases. For each method, we count the *unique* samples that only that method repairs (or breaks) while the other two do not. Tab. XI reports these counts, summed over the three ranks for each dataset and misclassification type.

While some misclassifications are repaired only by PRoViT$_{\mathrm{FT+LP}}$, far more inputs are broken only by it. It has the largest number of unique repairs in every row, but the counts are small. At the same time, it also produces by far the most unique breaks (48–199 per row), several times more than REPTRAN or ARACHNEW and far exceeding its own unique repairs. Thus, the few inputs that it alone repairs come at the cost of many more inputs that it alone breaks.

REPTRAN and ARACHNEW behave far more conservatively, repairing and breaking few samples uniquely. This is because they modify only a localized subset of the FFN weights, whereas PRoViT$_{\mathrm{FT+LP}}$ optimizes the entire FFN and therefore alters the model's behavior more aggressively. The only exception is TinyImg / *TGT-FN*, where REPTRAN records the most unique breaks (101 samples), driven by a single benchmark (rank 2) rather than the general trend.

C. Future Directions

While REPTRAN is promising as a practical repair method for Transformer models, it also leaves several important questions open for future research.

1) Applications to Large Models: In light of recent advances in large models such as LLMs, investigating how to adapt REPTRAN for such models is a promising direction for future research. These models are based on the Transformer architecture with billions of parameters and support widely used applications such as ChatGPT. As model size increases, data collection and retraining become increasingly costly; thus, repairing specific misbehaviors with limited data and time is especially valuable for large models. Since FFN components are essential even in state-of-the-art large models, REPTRAN is in principle applicable to them. However, it remains an open question whether FFN-only repair remains effective at this scale and whether the selection and search phases can be executed within a practical time.

2) Other Applications of Our Proposed Neuron Score: While our neuron score is primarily designed to guide model repair, it also holds potential for broader quality assurance tasks beyond repair, such as testing [30], [40]. Because the score reflects neuron behavior in Transformer models, it may facilitate the development of testing strategies specifically tailored to Transformer-based architectures, including LLMs. For example, test inputs can be synthesized to strongly activate neurons with high scores, helping to expose specific failures that may otherwise go undetected. In addition, existing test inputs can be prioritized based on the activation patterns of high-scoring neurons, enabling more efficient detection of misbehaviors under limited testing resources.

VII. THREATS TO VALIDITY

Internal Threats. REPTRAN involves several hyperparameters, such as the number of selected weights N_w, the balance coefficients p and α, and those for differential evolution. While these values were chosen based on prior studies or preliminary experiments, alternative settings may yield different outcomes. Although RQ3 and RQ4 investigate the effects of some key hyperparameters (i.e., N_w, p, and α), a comprehensive hyperparameter search is not included. Therefore, our findings may be partially dependent on specific configurations.

External Threats. We evaluated REPTRAN on two image classification datasets using ViT. To ensure coverage of diverse misclassifications, we constructed 18 fault benchmarks by varying misclassification types and ranks. Moreover, in all experiments, the samples used for repair and those used for evaluation were strictly separated to simulate realistic deployment scenarios. While these design choices enhance the generalizability of our findings, it remains unclear whether the results extend to other domains or architectures.

Construct Threats. We define successful repair as the correction of targeted misclassifications without breaking existing correct predictions, which aligns with prior work on DNN repair [5], [10], [16]. However, this definition may not fully capture broader quality attributes such as robustness, fairness, or safety, which are also important in practical deployments. Prior work [11], [31] suggests that the optimal repair method may vary depending on the specific quality attribute being targeted. Therefore, developing and evaluating repair methods for Transformer models from multiple perspectives beyond correctness remains an important direction for future work.

VIII. CONCLUSION

This paper introduced REPTRAN, a search-based repair method tailored to Transformer models. We evaluated REPTRAN on 18 fault benchmarks derived from CIFAR-100 and Tiny-ImageNet. The results showed that REPTRAN statistically significantly outperforms both random selection and ARACHNE in repair rate. It also slightly outperforms ARACHNEW, as indicated by the effect size. While REPTRAN incurs a higher break rate than some baselines, this rate remains below 5.7% and is justified by its substantially higher repair effectiveness. Our analysis in the discussion further showed that REPTRAN has complementary strengths to ARACHNEW and PRoViT. These findings highlight the effectiveness of REPTRAN and demonstrate that search-based repair can be

successfully extended beyond CNNs and RNNs to modern Transformer architectures.

IX. Data Availability

All data and code used in this study are available at the following repository: https://anonymous.4open.science/r/RepTran-replication-8E9E.

Acknowledgments

We gratefully acknowledge the financial support of: (1) Japan Society for the Promotion of Science (JSPS) for the KAKENHI grants (JP25K22845 and JP26H02500); (2) Japan Science and Technology Agency (JST) as part of Adopting Sustainable Partnerships for Innovative Research Ecosystem (ASPIRE), Grant Numbers JPMJAP2415 and JPMJAP2301; (3) JST-Mirai Program Grant No. JPMJMI20B8; (4) the Inamori Research Institute for Science (InaRIS Fellowship to Yasutaka Kamei).

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[2] Y. Hu, J. Yang, L. Chen, K. Li, C. Sima, X. Zhu, S. Chai, S. Du, T. Lin, W. Wang *et al.*, "Planning-oriented autonomous driving," in *Proc. of CVPR*, 2023, pp. 17 853–17 862.

[3] "OECD AI incidents and hazards monitor — autonomous vehicle incidents," https://tinyurl.com/2ezzrnft, accessed: 2026-04-09.

[4] B. Sun, J. Sun, L. H. Pham, and J. Shi, "Causality-based neural network repair," in *Proc. of ICSE*, 2022, pp. 338–349.

[5] J. Sohn, S. Kang, and S. Yoo, "Arachne: Search-based repair of deep neural networks," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 4, 2023.

[6] D. Li Calsi, M. Duran, X.-Y. Zhang, P. Arcaini, and F. Ishikawa, "Distributed repair of deep neural networks," in *Proc. of ICST*, 2023, pp. 83–94.

[7] D. Li Calsi, M. Duran, T. Laurent, X.-Y. Zhang, P. Arcaini, and F. Ishikawa, "Adaptive search-based repair of deep neural networks," in *Proc. of GECCO*, 2023, pp. 1527–1536.

[8] Z. Tao, S. Nawas, J. Mitchell, and A. V. Thakur, "Architecture-preserving provable repair of deep neural networks," in *Proc. ACM Program. Lang.*, vol. 7, no. PLDI. ACM New York, NY, USA, 2023, pp. 443–467.

[9] X. Sun, W. Liu, S. Wang, T. Chen, Y. Tao, and X. Mao, "Autoric: Automated neural network repairing based on constrained optimization," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 2, 2024.

[10] S. Nawas, Z. Tao, and A. V. Thakur, "Provable repair of vision transformers," in *International Symposium on AI Verification*. Springer, 2024, pp. 156–178.

[11] H. You, Z. Wang, Z. Dong, L. Mo, J. Zhao, and J. Chen, "A comprehensive study of deep learning model fixing approaches," *arXiv preprint arXiv:2512.23745*, 2025.

[12] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[13] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," *Master's thesis, University of Toronto*, 2009.

[14] Y. Le and X. Yang, "Tiny imagenet visual recognition challenge," *CS 231N*, vol. 7, no. 7, p. 3, 2015.

[15] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. of ICLR*, 2021.

[16] S. Tokui, S. Tokumoto, A. Yoshii, F. Ishikawa, T. Nakagawa, K. Munakata, and S. Kikuchi, "Neurecover: Regression-controlled repair of deep neural networks with training history," in *Proc. of SANER*, 2022, pp. 1111–1121.

[17] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. of ICNN*, vol. 4, 1995, pp. 1942–1948.

[18] M. Sotoudeh and A. V. Thakur, "Provable repair of deep neural networks," in *Proc. of PLDI*, 2021, pp. 588–603.

[19] M. Geva, R. Schuster, J. Berant, and O. Levy, "Transformer feed-forward layers are key-value memories," in *Proc. of EMNLP*, 2021, pp. 5484–5495.

[20] D. Dai, L. Dong, Y. Hao, Z. Sui, B. Chang, and F. Wei, "Knowledge neurons in pretrained transformers," in *Proc. of ACL*, 2022, pp. 8493–8502.

[21] P. Michel, O. Levy, and G. Neubig, "Are sixteen heads really better than one?" *Advances in neural information processing systems*, vol. 32, 2019.

[22] Y. Hao, L. Dong, F. Wei, and K. Xu, "Self-attention attribution: Interpreting information interactions inside transformer," in *Proc. of AAAI*, vol. 35, No. 14, 2021, pp. 12 963–12 971.

[23] Y. Li, J. Wang, X. Dai, L. Wang, C.-C. M. Yeh, Y. Zheng, W. Zhang, and K.-L. Ma, "How does attention work in vision transformers? a visual analytics attempt," *IEEE transactions on visualization and computer graphics*, vol. 29, no. 6, pp. 2888–2900, 2023.

[24] T. Tang, W. Luo, H. Huang, D. Zhang, X. Wang, X. Zhao, F. Wei, and J.-R. Wen, "Language-specific neurons: The key to multilingual capabilities in large language models," in *Proc. of ACL*, 2024, pp. 5701–5715.

[25] J. Niu, A. Liu, Z. Zhu, and G. Penn, "What does the knowledge neuron thesis have to do with knowledge?" in *Proc. of ICLR*, 2024.

[26] E. Mitchell, C. Lin, A. Bosselut, C. Finn, and C. D. Manning, "Fast model editing at scale," *arXiv preprint arXiv:2110.11309*, 2021.

[27] K. Meng, D. Bau, A. Andonian, and Y. Belinkov, "Locating and editing factual associations in gpt," *Advances in neural information processing systems*, vol. 35, pp. 17 359–17 372, 2022.

[28] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *Proc. of ICML*. PMLR, 2017, pp. 3319–3328.

[29] Y. Yuan, Q. Pang, and S. Wang, "Revisiting neuron coverage for dnn testing: A layer-wise and distribution-aware criterion," in *Proc. of ICSE*, 2023, pp. 1200–1212.

[30] Y. Huang, J. Song, Z. Wang, S. Zhao, H. Chen, F. Juefei-Xu, and L. Ma, "Look before you leap: An exploratory study of uncertainty analysis for large language models," *IEEE Transactions on Software Engineering*, vol. 51, no. 2, pp. 413–429, 2025.

[31] Y. Ishimoto, M. Kondo, L. Ma, N. Ubayashi, and Y. Kamei, "Repairs and breaks prediction for deep neural networks," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 4, 2025.

[32] O. N. Manzari, H. Ahmadabadi, H. Kashiani, S. B. Shokouhi, and A. Ayatollahi, "Medvit: a robust vision transformer for generalized medical image classification," *Computers in biology and medicine*, vol. 157, p. 106791, 2023.

[33] T. Nakagawa, S. Tokumoto, S. Tokui, and F. Ishikawa, "An experience report on regression-free repair of deep neural network model," in *Proc. of SANER*, 2023, pp. 778–782.

[34] C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer, "Genprog: A generic method for automatic software repair," *Ieee transactions on software engineering*, vol. 38, no. 1, pp. 54–72, 2011.

[35] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in statistics: Methodology and distribution*. Springer, 1992, pp. 196–202.

[36] N. Cliff, "Dominance statistics: Ordinal analyses to answer ordinal questions." *Psychological bulletin*, vol. 114, no. 3, p. 494, 1993.

[37] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data:should we really be using t-test and cohen's d for evaluating group differences on the nsse and other surveys," in *Proceedings of the Annual meeting of the Florida Association of Institutional Research*, 2006, pp. 1–33.

[38] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian journal of statistics*, pp. 65–70, 1979.

[39] W. H. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

[40] H. Guo, C. Tao, Z. Huang, and W. Zou, "White-box test input generation for enhancing deep neural network models through suspicious neuron awareness," *ACM Transactions on Software Engineering and Methodology*, 2025.