White-Box Test Input Generation for Enhancing Deep Neural Network Models through Suspicious Neuron Awareness

HONGJING GUO, CHUANQI TAO*, ZHIQIU HUANG*, and WEIQIN ZOU*, Nanjing University of Aeronautics and Astronautics, China

Deep Neural Network (DNN) testing has emerged as an effective way of uncovering erroneous behaviors in DNN models and further enhancing their performance. Research on test input generation has gained much attention from both researchers and practitioners, aiming to expose faults in models. The newly generated inputs subsequently serve as additional training instances for model refinement through retraining. Existing approaches generate test inputs by optimizing an objective function based on testing metrics such as neuron coverage and property-related metrics, and the gradient of the objective is used to perturb seed inputs. However, these approaches pay limited attention to the model's decision logic, particularly the erroneous decision patterns learned during training. Furthermore, they primarily focus on detecting faults without considering the diversity of detected misbehaviors, which limits the models' ability to learn diverse features through retraining. To address these limitations, this paper introduces SUNTest, a novel test input generation approach designed to detect diverse faults and enhance the robustness of DNN models. SUNTest focuses on erroneous decision-making by localizing suspicious neurons responsible for misbehaviors through the execution spectrum analysis of neurons. To guide input mutations toward inducing diverse faults, SUNTest designs a hybrid fitness function that incorporates two types of feedback derived from neuron behaviors, including the fault-revealing capability of test inputs guided by suspicious neurons and the diversity of test inputs. Additionally, SUNTest adopts an adaptive selection strategy for mutation operators to prioritize operators likely to induce new fault types and improve the fitness value in each iteration. Experiments conducted on eight DNN models demonstrate the effectiveness of SUNTest in fault localization and test input generation. It outperforms existing test input generators in the number of detected faults, uncovering up to 80.9 more distinct fault types. In terms of model enhancement, SUNTest increases the average accuracy improvement by up to 8.04% compared to baseline approaches.

CCS Concepts: • **Software and its engineering**; • **Software testing and debugging**; • **Neural networks**;

Additional Key Words and Phrases: Deep Neural Network Testing, Test Input Generation, Fault Localization, Model Retraining

1 INTRODUCTION

Deep Neural Network (DNN) models are progressively being integrated into intelligent software systems and are pervasive in safety-critical domains, such as medical diagnosis [16], autonomous driving [11], and aircraft collision avoidance [34]. As the core components of intelligent software, DNN models facilitate advanced functionalities through perception, prediction, and decision-making. Despite their tremendous success in various

*The authors are with the Key Laboratory for Safety-Critical Software Development and Verification, Ministry of Industry and Information Technology (Nanjing University of Aeronautics and Astronautics), Nanjing, China, 211106. They are with the Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing, China, 210023. Chuanqi Tao is also affiliated with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China, 210023. Chuanqi Tao is the corresponding author.

Authors' address: Hongjing Guo, guohongjing@nuaa.edu.cn; Chuanqi Tao, taochuanqi@nuaa.edu.cn; Zhiqiu Huang, zqhuang@nuaa.edu.cn; Weiqin Zou, weiqin@nuaa.edu.cn, Nanjing University of Aeronautics and Astronautics, No. 29 Jiangjun Avenue, Jiangning District, Nanjing, China, 211106.

ACM 1557-7392/2025/5-ART
https://doi.org/10.1145/3736305

domains, numerous severe accidents have arisen due to inherent quality issues within DNN models [3, 63]. Therefore, the imperative to ensure and enhance the quality of DNN models has become critical and urgent. DNN testing has emerged as an effective way of uncovering erroneous behaviors and boosting the trustworthiness of models. Inspired by traditional software testing approaches, the software engineering research community has made significant efforts to propose novel approaches specific to DNN models for test input generation [9, 25, 43, 65, 68, 75, 83], test input selection [10, 18, 50, 61], and test adequacy analysis [21, 36, 37, 47, 74, 78]. In this study, we mainly focus on the topic of test input generation.

Recent DNN testing studies on test input generation aim to produce inputs specifically designed to uncover the misbehaviors of DNN models. These inputs are then incorporated as additional training instances to enhance model performance through retraining. Metric-guided fuzzing (MGF) is the mainstream technique for generating test inputs. At a high level, fuzz testing (*a.k.a*, fuzzing) mutates existing test inputs to generate new ones, with the objective of detecting new faults by exploring large search spaces [52]. Existing MGF techniques [9, 25, 43, 68] address the test input generation problem using an optimization framework designed to expose errors while maximizing a specific testing metric in a limited testing budget. The optimization objective is formulated based on testing metrics such as coverage and property-related metrics (*e.g.*, neuron coverage metric [43], robustness-oriented metric [9, 68]). The gradient of the objective function is used to perturb the input, directing it towards an increase in the objective value. For example, approaches like DLFuzz [24] and ADAPT [43] utilize the gradients of internal neurons within DNNs to perturb inputs. The goal is to generate fault-revealing test inputs while achieving high neuron coverage. RobOT [68] adopts robustness-oriented testing metrics for designing its optimization objectives. While existing MGF approaches have demonstrated high effectiveness in generating fault-inducing test inputs, they still suffer from the following limitations.

Previous approaches have paid limited attention to generating test inputs based on an understanding of the model's decision-making mechanisms, particularly its erroneous decision logic. In traditional software testing, structural test input generators aim to produce test cases that maximize coverage targets, such as branch and statement coverage [55]. For branch coverage, control dependencies that characterize program logic are exploited to search for test cases that meet coverage targets. Similarly, existing MGF approaches select neurons within DNN models for generating test inputs by adding gradients to the original inputs, aiming to increase coverage [25, 43]. However, unlike code-level coverage for traditional software programs, neuron coverage does not necessarily fully exercise the implicit logic embedded in DNNs. Recent studies [26, 44] also have highlighted that neuron coverage exhibits a limited correlation with the decision logic of DNN models. Since the decision-making process of DNN models relies on internal neurons [21, 74], identifying which neurons are crucial to decision-making remains a challenge for testers, particularly those responsible for erroneous decision logic. The absence of white-box analysis makes it difficult to find which critical components have been overlooked during debugging and testing. Existing neuron selection strategies designed by approaches like DLFuzz [25] and ADAPT [43] often fail to prioritize neurons that are crucial for inducing misbehaviors. Given the vast search space of possible test inputs, this limitation constrains the generation of test inputs that specifically target fault-inducing neurons, thereby hindering the ability to stress the model to expose faults. Therefore, the lack of targeted white-box testing approaches not only hinders the understanding of DNN behaviors but also limits the effectiveness of test input generation in addressing the root causes of faults.

Second, previous approaches do not consider the diversity of detected incorrect behaviors of DNN models. Intuitively, a highly diverse set of test inputs can better explore the fault space, thereby increasing the fault detection capability of the input set. Existing MGF approaches employ gradient-guided strategies to produce pixel-level perturbations. However, these strategies face challenges due to gradient vanishing. When the gradient is small, the generated test inputs tend to be highly similar [83]. It limits the capability of DNN models to learn more diverse features during the retraining process. Additionally, these approaches [25, 43] rely solely on the number of inputs that induce errors to define the fault detection capability of a test input set. It is misleading

because some test inputs may share the same types of faults. Gradients do not convey information related to fault diversity, which may hinder the effectiveness of uncovering diverse fault types in DNN models. This limitation is particularly pronounced when developers encounter numerous test inputs exhibiting duplicated fault types, making it difficult to sufficiently repair DNN models.

To address the aforementioned challenges, this paper proposes a novel test input generation approach, termed **SU**spicious **N**euron-aware **Test** input generation (SUNTest). The approach is designed with two primary goals: 1) to characterize erroneous decisions in DNN models by localizing suspicious neurons, and 2) to generate test inputs that induce diverse faults with a focus on erroneous decision-making. Specifically, SUNTest begins by determining dynamic neuron activation thresholds through an analysis of neuron output distributions. It establishes an execution spectrum for each neuron by monitoring its behaviors during the execution of training inputs. It then localizes the neurons responsible for erroneous decisions, referred to as suspicious neurons. Following this, SUNTest generates new test inputs by iteratively mutating seed inputs using various domain-specific mutation operators. To guide the mutation process, SUNTest introduces a hybrid fitness function that incorporates two feedback derived from neuron behaviors: 1) error-revealing capability feedback, which searches for test inputs that could increase the outputs of suspicious neurons, and 2) input diversity feedback, which favors test inputs exhibiting high diversity. Newly mutated test inputs that increase the fitness value are adopted as seeds for further mutation iterations. Additionally, an adaptive operator selection strategy that learns from historical test execution results of mutation operators is introduced. In each iteration, it prioritizes selecting a mutation operator that is likely to trigger a new fault type and improve the fitness value. Finally, the newly generated test inputs serve as additional training instances to enhance the model through retraining.

To evaluate the effectiveness of SUNTest, an empirical evaluation is conducted on four widely used datasets and eight carefully designed classification models. Adversarial attack strategies are employed to simulate diverse testing scenarios with varying data distributions. The experimental results demonstrate that SUNTest outperforms existing fault localization approaches. It surpasses state-of-the-art test input generators, increasing the number of detected faults by up to 310.5. The increase in the number of triggered fault types ranges from 29.2 to 80.9. In terms of model enhancement, SUNTest increases the average accuracy improvement by up to 8.04% compared to baseline approaches, achieving an improvement rate of up to 195.8%. Additionally, both the guidance from suspicious neurons, the adaptive mutation operator selection strategy, and the hybrid fitness function contribute to the effectiveness of SUNTest. The replication package can be found in our online artifact [1].

The contributions of this paper could be summarized as follows:

- We propose SUNTest, a novel test input generation approach for DNN models guided by the analysis of erroneous decisions. SUNTest is designed to detect diverse faults and improve the robustness of DNN models.
- By employing dynamic activation thresholds based on neuron output distributions, the neuron spectrum is introduced to localize suspicious neurons responsible for erroneous decisions in DNN models.
- A hybrid feedback mechanism based on neuron behaviors and an adaptive selection strategy for mutation operators are designed to guide mutations toward generating error-revealing test inputs capable of triggering diverse fault types.
- An Empirical evaluation is conducted to investigate the effectiveness of SUNTest. The results demonstrate its effectiveness in localizing suspicious neurons. SUNTest significantly outperforms other test input generation approaches in terms of triggering diverse faults and enhancing model robustness.

This paper extends our previous work [24] published in ISSRE 2023, which introduces suspicious neurons for test input selection. The prior work, called MOON, focuses on selecting a subset of test inputs from an unlabeled test set for model retraining. SUNTest mutates seed inputs to uncover errors and improve model robustness through retraining. Both approaches leverage the concept of suspicious neurons to characterize erroneous

decisions made by DNNs and evaluate the potential of inputs to expose errors based on neuron behaviors. The key differences between the two approaches lie in their downstream tasks and methodologies. MOON transforms test input selection into a search problem, employing a multi-objective optimization algorithm to identify test inputs that maximize two objectives: suspicious neuron activation and input diversity. SUNTest adopts a fuzzing-based framework for test input generation by mutating seed inputs. It formulates the mutation process as a Markov Chain problem, designing a hybrid fitness function based on neuron behaviors to determine whether a mutated input can be used in subsequent iterations. Additionally, we integrate MOON to sample initial test inputs from the input corpus as seed candidates for the fuzzing-based test input generation process. This integration constitutes an important step in SUNTest's overall test generation strategy.

The rest of this paper is organized as follows: Section 2 introduces the background knowledge of this work. Section 3 presents details of the proposed approach. Section 4 outlines experimental settings. Section 5 provides experimental results and analysis. In Section 6, implications of this work and threats to validity are discussed. Section 7 presents the related work, and section 8 concludes this paper.

2 BACKGROUND

2.1 Spectrum-Based DNN Fault Localization

DNN models consist of stacked, non-linear layers with highly interconnected neurons, rendering the model logic opaque to human understanding. The internal complexity and opacity of DNN models pose challenges in the context of fault localization. The term 'fault' is adopted to describe situations where the DNN model generates an output, but the actual result deviates from the expected outcome. Factors such as poor-quality training data, an inadequate training process, or suboptimal hyperparameters can contribute to the occurrence of such faults. Since the decision-making process of DNNs relies on neuron outputs, previous fault localization techniques for DNNs focus on pinpointing neurons responsible for erroneous behaviors. These techniques draw an analogy between neurons responsible for erroneous behaviors in DNNs and faulty statements in traditional programs. The paradigm of Spectrum-Based Fault Localization (SBFL) [32, 85], originally developed for traditional software programs, has been extended to DNN models. SBFL techniques for traditional software take a faulty program and a set of test cases as input, producing a ranked list of suspicious program elements as output. It begins by analyzing the frequency of program elements executed (covered) by both passing and failing test cases to construct the program's execution spectrum [73]. Using a risk evaluation formula (*i.e.*, suspiciousness measure), SBFL then calculates suspicious scores for program elements, indicating their likelihood of being faulty. Techniques such as Tarantula [33], Ochiai [2], and DStar [72] follow the above paradigm but utilize different risk evaluation formulas to compute suspicious scores of program elements.

For spectrum-based DNN fault localization, each neuron within DNN models is treated as a distinct component. Since DNN decisions are collectively determined by all neurons within the model [78], the concepts of 'executed' and 'non-executed' are not applicable. To capture the execution state of neurons, continuous neuron outputs are converted into discrete states, *i.e.*, activated or inactivated [15, 58]. Given a DNN model D, the spectrum of each neuron n is defined as a tuple $N_{SP} = (A_n^{as}, A_n^{af}, A_n^{is}, A_n^{if})$. Specifically, attributes A_n^{as} and A_n^{af} denote the number of times neuron n was active when the model D made correct and incorrect predictions. Attributes A_n^{is} and A_n^{if} signify the number of times neuron n was inactivated when D made a successful and failed decision. SBFL formulas for traditional software are adapted to measure the correlation between each neuron and the detected erroneous behaviors in the context of DNN models. Neurons with higher suspiciousness scores are presumed to have a stronger correlation with erroneous behaviors. This paper tailors SBFL techniques to establish the execution spectra for neurons by employing dynamic neuron activation thresholds, thereby facilitating the localization of suspicious neurons.

2.2 Fuzzing-based Test Input Generation

In the Software Engineering community, the generation of test inputs for DNN models has been introduced as a means to produce artificial inputs. These inputs are adopted to assess whether a DNN model can generalize beyond the initial training set. Recent research in DNN testing focuses on generating test inputs that are specifically designed to induce potential misbehaviors of DNN models. Formally, given a non-triggering input x as a seed, the goal is to generate a new input x' by applying domain-specific perturbations to the seed, such that x' induces a different prediction output from the DNN model compared to the expected output. The expected output of x' is assumed to be the same as the output of x. By incorporating the newly generated inputs into the training process, the DNN model learns to handle a wider variety of scenarios, thereby increasing its accuracy and robustness. This iterative process of testing and retraining is fundamental to the development of reliable DNN models.

Recent studies propose MGF approaches for generating test inputs, primarily aimed at exposing errors while optimizing specified testing metrics within a limited testing budget. Previous MGF approaches [25, 43, 68] tackle the challenge of generating test inputs through an optimization framework. The optimization objective is formulated based on specific testing metrics (*e.g.*, coverage metrics, model property-correlated metrics, uncertainty metrics). Formally, let $M = \{m_1, m_2, ..., m_k\}$ be the set of testing metrics. The goal is to find a set of test inputs $T = \{t_1, t_2, ..., t_n\}$ that maximizes the optimization objective function $Q_M(T)$:

$$max \quad Q_M(T) = \bigcup_{t_j \in T} \sum_{i=1}^{k} \omega_i O\left(m_i, t_j\right). \tag{1}$$

Each $O\left(m_i, t_j\right)$ denotes the value of test input t_j with respect to the testing metric m_i, and ω_i represents the weight of metric m_i. The gradient of the objective is employed to perturb the seed input, directing it towards an increase in the objective value. Particularly, Coverage-Guided Fuzzing (CGF) is one of the MGF techniques that adopts coverage metrics to formulate the optimization objective. It aims to generate test inputs that are likely to increase the DNN coverage. This paper evaluates the effectiveness of SUNTest through a comparative analysis with state-of-the-art MGF approaches.

The primary objective of this formulation is not to simultaneously maximize the test objectives and minimize the size of the test suite T. Instead, existing approaches aim to generate test inputs under predefined stopping criteria, with the goal of maximizing the objective functions. While optimizing both test objectives and test suite size is feasible, it remains an underexplored area for DNN testing. Investigating this trade-off presents a promising direction for future research.

3 APPROACH

3.1 Overview

Figure 1 provides an overview of SUNTest, which comprises two main components: suspicious neuron localization and test input generation. Initially, SUNTest determines the dynamic neuron activation threshold by analyzing neuron output distributions. It constructs the neuron spectrum by monitoring neuron behaviors during the execution of both correctly and incorrectly classified training inputs. Subsequently, SBFL formulas are employed to identify suspicious neurons responsible for erroneous behaviors of DNN models. Suspicious neurons are utilized as guidance to generate error-revealing test inputs.

SUNTest generates test inputs through multiple iterations of mutations. It selects inputs from the original testing set as seeds. In each iteration, an adaptive mutation operator selection strategy is employed to generate a new test input by mutating the seed. The mutated test inputs are fed into the DNN model to observe neuron behaviors and obtain test execution results. To guide the mutation process, SUNTest utilizes a hybrid fitness function that combines error-revealing capability feedback and input diversity feedback. Guided by suspicious

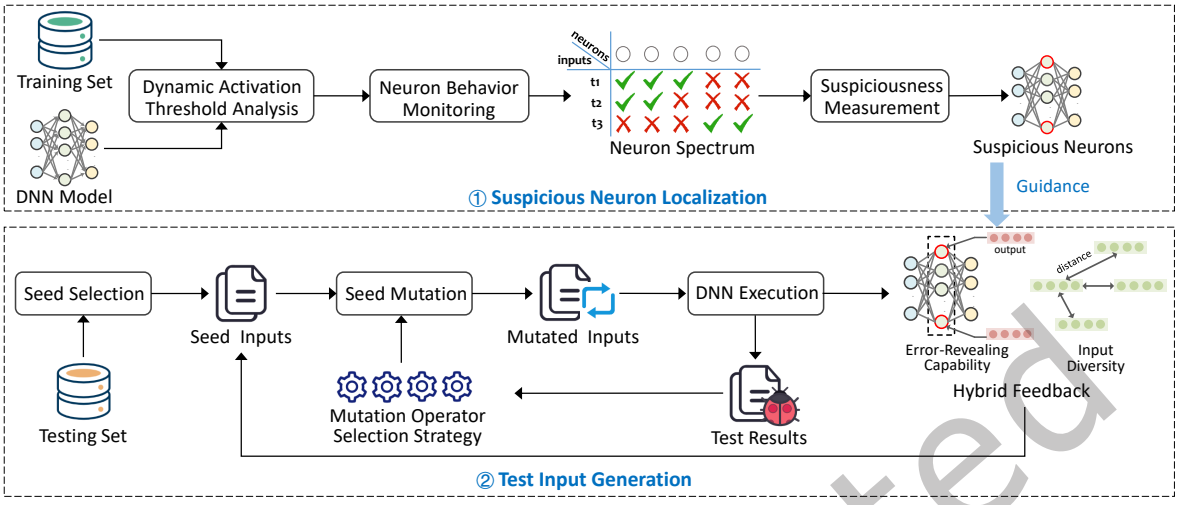

Fig. 1. An Overview of SUNTest

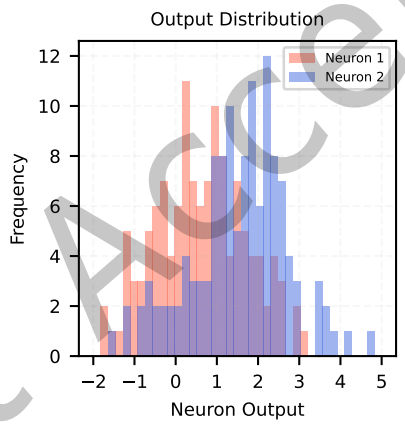

Fig. 2. Output Distribution of Two Neurons

neurons, this hybrid feedback mechanism is designed to favor test inputs likely to trigger errors and exhibit high diversity. These 'valuable' test inputs are adopted as seeds for further mutation iterations. The test execution results of each mutation, particularly the frequency of triggering new fault types and improving fitness values, are used to update the operator selection strategy.

3.2 Suspicious Neuron Localization

To capture the behaviors of neurons, SUNTest establishes a neuron spectrum for each neuron based on the dynamic neuron activation threshold. Neurons that respond violently to misclassifications are identified and categorized as suspicious neurons.

3.2.1 Dynamic Neuron Spectrum Analysis. Followed by the definition in Section 2.1, SUNTest establishes the neuron spectrum by inputting both passed (correctly classified by the model) and failed (misclassified by the model) inputs into the DNN model. Since the decision logic of the DNN model is determined by the training set, SUNTest establishes the neuron spectrum through the execution of training inputs. It is designed to identify neurons responsible for erroneous decisions and learned errors during the training phase.

To capture the execution states of neurons, continuous neuron outputs are converted into discrete states, *i.e.*, activated or inactivated [15, 58]. A neuron is considered activated if its output exceeds a predefined threshold (*e.g.*, 0.5 or 0.75). Nevertheless, assigning neurons as either activated or inactivated based on a fixed activation threshold fails to capture the variations in output among different neurons. To illustrate this, we randomly select two neurons from the investigated DNN model (MNIST-ConvNet) and generate their corresponding output distributions by executing training inputs, as depicted in Figure 2. In the figure, the x-axis represents the output values, while the y-axis represents the frequency of the involved inputs. The orange and blue areas represent the output distributions of the two neurons, respectively. The darker areas indicate the regions where these distributions overlap. Noticeable differences are evident in the output distributions, particularly in the high-frequency regions. Intuitively, each neuron's output follows its statistical distribution. Therefore, to capture the variation in neuron behaviors, SUNTest is tailored to the statistical properties of each neuron's outputs, rather than relying on a fixed threshold. SUNTest employs dynamic activation thresholds based on the output distributions of neurons. Specifically, it creates a histogram HT_n for each neuron, where the x-axis denotes the output value and the y-axis represents the number of involved inputs. Initially, by feeding the training set T into D, SUNTest computes the outputs of neuron n across all training inputs and divides the output range into b equal intervals. It then determines the proportion of training inputs within each output interval. Finally, SUNTest derives the activation threshold for neuron n as the average value of the top-K most frequent output intervals. The dynamic activation threshold for neuron n is defined as follows:

$$TS_n = \frac{\sum_{i=1}^{K} \overline{\psi_i^l + \psi_i^u}}{K}, \tag{2}$$

where ψ_i^l and ψ_i^u denote the lower and upper bound of output interval i, respectively. In the experiments, b is set to 1000, and K is set to 100. The activation threshold TS_n represents the major behaviors of the neuron n. As depicted in Figure 2, the clustered output values capture the major behaviors of neurons, whereas neuron outputs lying in low-frequency regions characterize corner-case behaviors. Moreover, since layers closer to the output layer extract high-order features [10, 21], SUNTest selects the penultimate fully connected layer as the target layer for constructing the spectrum for each neuron.

3.2.2 Neuron Suspiciousness Measurement. A risk evaluation formula assigns a suspiciousness score to each neuron based on four attributes within the neuron spectrum. The suspiciousness score for each neuron reflects its accumulated response to both correctly and incorrectly predicted instances from the training set. This score serves as an indicator of how much a neuron contributes to the model's overall decision-making errors. SUNTest incorporates well-established SBFL formulas, namely Tarantula [33], Ochiai [2], and DStar [72]. These formulas have been widely adopted in the domain of automated software debugging [32, 73, 85]. In this section, we have omitted the formulation of risk evaluation formulas. Interested readers are referred to the literature [15, 24] for detailed definitions of these formulas as applied in DNN fault localization. In particular, SUNTest adopts a fixed value of 3 for DStar's variable $*$, following recommendations from the literature [57].

Once suspiciousness scores are computed for neurons in the target layer, they are ranked in descending order. The top λ percentile of neurons is selected as suspicious neurons. SUNTest calculates suspiciousness scores by treating each neuron as a basic computational unit. Model misbehaviors are often linked to multiple neurons. The hyperparameter λ is employed to adjust the number of selected suspicious neurons within a target layer. This

flexibility allows practitioners to aggregate multiple suspicious neurons within a layer, linking them to observed model misbehaviors. By doing so, it facilitates the identification of neuron combinations within a layer that may collectively contribute to these misbehaviors.

3.3 Test Input Generation Based on Suspicious Neurons

Table 1. Mutation Operators Adopted in SUNTest

Pixel-Level	Description	Image-Level	Description
Random Pixel Change	Randomly change the values of pixels to arbitrary in [0, 255]	Translation	Moves the pixels of the image for a certain distance
Add Salt & Pepper Noise	Randomly convert the values of pixels into 255 or 0	Brightness Adjustment	Adjust the brightness to simulate illumination changes
Add Gaussian Noise	Add a randomly generated Gaussian-distributed noise	Rotation	Rotate the image for a certain angle
Add Multiplicative Noise	Add a randomly generated Multiplicative nose	Median Blurring	Blur the image using a random median filter
		Average Blurring	Blur the image using a random average filter
		Gaussian Blurring	Blur the image using a random Gaussian filter

Suspicious neurons, accountable for misclassifications, reflect the erroneous decision logic of a DNN model. Leveraging suspicious neurons as a guide for generating test inputs enables the induction of the model's incorrect decision logic. Existing studies often overlook the diversity of the generated test inputs, resulting in limited capability to diversify detected fault types. Therefore, SUNTest generates test inputs guided by outputs of suspicious neurons, which aims to trigger the faulty decision logic within DNN models. Besides, to diversify the detected behaviors, SUNTest also takes into account the diversity of mutated inputs. Formally, given a non-triggering test input x as a seed, SUNTest aims to produce a follow-up test input x such that the top-1 label predicted by the DNN model D differs from that of x. This is achieved by applying a mutation operator O_p to x, resulting in $x' = x + O_p$. The mutation process is guided by a hybrid fitness function and a mutation operator selection strategy. The goal is to generate error-revealing test inputs that can detect various types of faults.

Since mutation operators are domain-specific, this paper primarily focuses on image classification models. Existing MGF techniques [9, 25, 43, 68] typically perturb a small set of pixels in an image to generate a test input. SUNTest not only uses pixel-wise changes but also incorporates image-level mutations for test input generation. The primary focus of SUNTest is not on introducing new mutation operators, but rather on adaptively adjusting the priority of operators in the queue based on the results of test execution. The detailed mutation operators adopted in this study are listed in Table 1. These mutation operators are widely utilized in simulating real-world scenarios in recent studies [65, 66, 77].

3.3.1 Test Input Generation Algorithm. Algorithm 1 depicts the details of the neuron behavior-guided test input generation algorithm. The algorithm takes the DNN model under test D, the hybrid fitness function F, a pool of mutation operators L, and iteration times max_iter as inputs. The output of the algorithm is the generated test input set, denoted as R, capable of triggering errors. Within the pool of mutation operators, each operator maintains a reward score that signifies its priority for selection during the iteration.

The algorithm maintains a seed list S that stores 'valuable' seed inputs. Initially, the algorithm randomly selects a seed input x from the seed_list S (lines 3-4). f_{max} stores the maximum fitness value among all generated test inputs. It is initially set to the fitness value of x according to the hybrid fitness function (line 5). The mutation operator $latest\_O_p$ is initialized as $\varnothing$. Subsequently, the algorithm generates test inputs through multiple iterations of mutations, as indicated by the inner loop in lines 7-25. If the mutation operator is $\varnothing$, the algorithm randomly selects an operator from the pool of operators to initialize $latest\_O_p$ (lines 8-9). Otherwise, the algorithm adopts an operator selection strategy to choose an operator O_p (lines 10-11). The adaptive operator selection process favors an operator that is more likely to make the mutated test input trigger diverse errors. The selected operator is then applied to mutate the input x, generating a new test input x (line 13). The fitness value of x' is obtained

Algorithm 1 Neuron Behavior-Guided Test Input Mutation for DNN Models

Input: An DNN model under test D, the hybrid fitness function F, the seed list S, the mutation operator pool L, iteration times max_iter
Output: Generated test input set R

1: $R \leftarrow \emptyset$
2: **while** not_empty(S) **do**
3: $x \leftarrow obtain\_seeds(S)$ ▷ randomly pick a seed from the list of seeds S
4: $S \leftarrow S \setminus x$
5: $f_{max} \leftarrow obtain\_fitness(D, x, F)$
6: $latest\_O_p \leftarrow \emptyset$
7: **for** $t = 1$ to max_iter **do**
8: **if** $latest\_O_p == \emptyset$ **then**
9: $latest\_O_p \leftarrow initialize(L)$ ▷ randomly select an operator from the pool
10: **else**
11: $O_p \leftarrow find\_operator(latest\_O_p, P)$ ▷ MCMC-guided mutation operator selection
12: **end if**
13: $x' \leftarrow O_p(x)$ ▷ mutate the input x with the operator
14: $f_{x'} \leftarrow obtain\_fitness(D, x', F)$
15: **if** $\xi(x) \neq \xi(x')$ **then** ▷ check whether the model outputs inconsistent results
16: $R \leftarrow R \cup x'$
17: **else**
18: **if** $f_{x'} > f_{max}$ **then** ▷ check whether the objective value is increased
19: $f_{max} \leftarrow f_{x'}$
20: $x \leftarrow x'$
21: $latest\_O_p \leftarrow O_p$
22: **end if**
23: **end if**
24: $update(O_p)$ ▷ update the reward score of the operator
25: **end for**
26: **end while**
27: **return** R

using the function $obtain\_fitness()$ (line 14). Next, if the DNN model produces different prediction results for x and x, x is added to the set R (lines 15-16). The algorithm determines whether to update f_{max} by comparing the fitness value f_x with f_{max} and selects the input with a higher fitness value for the next iteration (lines 18-22). Finally, the reward score of the operator O_p is updated at the end of each iteration based on the test execution results, including the error-revealing capability, the degree of fitness value enhancement, and the frequency of the operator being selected (line 24).

3.3.2 Hybrid Feedback Mechanism. SUNTest introduces a hybrid feedback mechanism designed to determine whether the newly mutated input x' could be used as a seed input for subsequent mutation iterations. Based on neuron behaviors, it formulates a hybrid fitness function that integrates error-revealing capability and input diversity measurement as the fuzzing feedback. It aims to favor inputs that are more prone to inducing erroneous behaviors and triggering diverse behaviors of DNN models.

Error-Revealing Capability Feedback. In the case of test inputs capable of inducing errors, the erroneous decisions made by DNN models are predominantly influenced by suspicious neurons. Therefore, SUNTest formulates the fitness function $f(x)$ to favor inputs capable of activating suspicious neurons to a large extent,

aiming to induce misclassifications in subsequent iterations. This is achieved by maximizing the outputs of a dynamically combined set of suspicious neurons. The fitness function $f(x)$ is formulated as follows:

$$f(x) = \sum_{k=1}^{N_k} [(b_k | h_k) \phi_k(x)]. \tag{3}$$

N_k denotes the number of suspicious neurons within the target layer, where $\phi_k(x)$ represents the output of the k-th suspicious neuron with respect to an input x. The vector b consists of N_k elements, each corresponding to a suspicious neuron, and it is initialized with all elements set to 0. SUNTest dynamically combines suspicious neurons by introducing a random vector h, which indicates the positions of randomly selected suspicious neurons. Specifically, the vector h is defined as $h = RANDOM_{(0,1)}(b)$, where $RANDOM_{(0,1)}(b)$ randomly generates a Boolean vector of the same size as b, with each element h_k randomly assigned a value of either 1 or 0. The sum of elements in h, denoted as $\sum h$, is given by $\sum h = ROUND(N_k \times q)$, where $ROUND()$ rounds the result to the nearest integer. In this context, SUNTest sets q to 0.8. Given that b_k and h_k are elements of the vectors b and h, respectively, the symbol '|' in this case represents the logical OR operation. The condition $b_k | h_k = 1$ holds true if at least one of b_k or h_k is equal to 1. To illustrate, consider a simple case where $N_k = 5$ and the vector $b = [0, 0, 0, 0, 0]$, representing five suspicious neurons in the target layer. The random vector h is generated as $h = [1, 1, 1, 1, 0]$, with 4 elements set to 1, as determined by the formula $5 \times 0.8 = 4$. Consequently, the condition $b_k | h_k = 1$ results in $[1, 1, 1, 1, 0]$, as the logical OR between corresponding elements of b and h results in 1 whenever at least one of b_k or h_k is 0.

Input Diversity Feedback. Diverse test inputs, exhibiting variations from one another, could reveal diverse behavior patterns within the DNN model, enabling the detection of a broad range of fault types. Besides, improving the effectiveness of retraining by incorporating a more varied set of inputs, rather than utilizing a set of similar inputs, allows the DNN model to learn diverse features. Therefore, SUNTest maximizes a fitness function designed to measure the dissimilarity between an input and the previously generated error-revealing inputs. Specifically, SUNTest computes the distance between the individual input x and the inputs within the set R, where R represents the set of previously generated error-revealing test inputs. The decision behaviors of DNN models, when confronted with inputs, are characterized by neuron outputs. Consequently, the distance is calculated based on neuron outputs. The fitness function, denoted as $g(x)$, is formulated as follows:

$$g(x) = min_{y \in R, y \neq x} \left\| \sum_{i}^{N} \phi_i(x) - \sum_{i}^{N} \phi_i(y) \right\|_p. \tag{4}$$

N denotes the neurons in the target layer. The cumulative outputs of all neurons in the target layer associated with the input x are denoted as $\sum_i^N \phi_i(x)$. $\|\cdot\|_p$ denotes the L^p-norm. SUNTest employs the Euclidean distance to evaluate the input diversity, where p is set to 2 in the experiments.

Hybrid Fitness Function. The hybrid fitness function $F(x)$ for an input is defined as:

$$F(x) = \begin{cases} f(x) & R = \varnothing \\ f(x) + g(x) & R \neq \varnothing. \end{cases} \tag{5}$$

In the process of generating test inputs, when error-revealing test inputs have not been generated, the set R remains empty. In such instances, the fitness function $F(x)$ is solely determined by $f(x)$. In contrast, when $R(x)$ is non-empty, the fitness function $F(x)$ is defined as a combination of $f(x)$ and $g(x)$.

3.3.3 Selection Strategy of Mutation Operators. Various operators induce different types of mutation in the input, guiding the seed input towards triggering diverse behaviors in DNN models. In each iteration, SUNTest

employs an adaptive selection strategy that learns from historical test execution results to prioritize an operator for mutating the seed input. This strategy aims to favor an operator that is likely to produce the next mutated test input x' with a higher fitness value than x, while also being capable of introducing new fault types. To achieve this goal, the mutation operator selection process is formulated as a Markov Chain Monte Carlo (MCMC) problem [35]. MCMC is utilized to sample from a probability distribution by constructing a Markov Chain that converges to the desired distribution. This process can be modeled as a stochastic process $\{O_{p_0}, O_{p_1}, O_{p_2}, ..., O_{p_t}\}$, where each state O_{p_i} signifies the selected operator at the i-th iteration. The selection of $O_{p_{i+1}}$ from all possible operators within the pool is solely determined by O_{p_i}. SUNTest employs the Metropolis-Hastings (MH) algorithm [35] to guide the selection of mutation operators.

During each iteration, each operator is assigned a reward score according to the reward function. Suppose O_{p_i} is the selected operator at the i-th iteration of mutation. To determine which operator to use for the $(i+1)$-th iteration, operators are ranked in descending order based on their reward scores. To avoid repeatedly selecting the highest-ranked operator in subsequent iterations, which may produce duplicate test inputs, an acceptance probability for the state transition is introduced. This acceptance probability determines whether to execute the state transition, *i.e.*, moving from the current state to the next state. It ensures that each operator has a chance of being selected, with higher-ranked operators having a greater likelihood. The acceptance probability for $O_{p_{i+1}}$ given O_{p_i} is defined as follows:

$$P\left(O_{p_{i+1}}|O_{p_i}\right) = min(1, (1-p)^{r_{i+1}-r_i}), \tag{6}$$

where p represents the multiplicative inverse of the number of mutation operators in the pool. r_i and r_{i+1} denote the positions of O_{p_i} and $O_{p_{i+1}}$ in the priority list. If $O_{p_{i+1}}$ is ranked higher than O_{p_i} (*i.e.*, $r_{i+1} < r_i$), then $O_{p_{i+1}}$ is accepted. Otherwise, $O_{p_{i+1}}$ still has a certain probability $(1-p)^{r_{i+1}-r_i}$ to be accepted. Before the end of each iteration, the reward scores of operators are dynamically adjusted based on the information gathered from historical test execution results.

Reward Function. The reward score of an operator is gauged by its capability of generating test inputs with diverse fault types and enhancing the fitness value. Specifically, the reward function is designed based on two factors: 1) The frequency of triggering new fault types. How many distinct fault types have been detected based on the operator? 2) The frequency of improving the fitness value. How often the fitness value has been enhanced by the operator? The reward function is formulated as follows:

$$r\left(O_p\right) = \frac{\delta_i}{\eta_i + \varepsilon} \times \frac{\omega_i}{\eta_i + \varepsilon}. \tag{7}$$

The symbol i denotes the number of iterations. η_i represents the total number of times that the operator has been selected. To prevent the case of division by zero when η_i equals zero, ε is set to 1e-5. δ_i represents the number of non-redundant fault types detected by the operator, while ω_i signifies the frequency with which the fitness value has been enhanced by the operator.

4 EXPERIMENTAL SETTINGS

SUNTest is implemented on an Ubuntu 18.04 server, using Python (v3.8) as the programming language. The open-source Machine Learning frameworks employed in this study are Keras (v2.5.0rc0) and TensorFlow (v2.5.0). The geatpy library (v2.6.0) is utilized to implement the search-based test input selection algorithm. All experiments were conducted on a machine with an NVIDIA GTX 3090 GPU, an Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz processor, and 160 GB of memory.

4.1 Research Questions

In the experimental evaluation, we aim to answer the following five research questions.

- **RQ1: Can SUNTest localize suspicious neurons effectively?**

Since suspicious neurons contribute to erroneous decision results, reversing the weights associated with these neurons could potentially mitigate error propagation to subsequent layers [82], thereby improving the accuracy of DNN models. To evaluate the effectiveness of SUNTest in localizing suspicious neurons, we reverse the sign of the weights of these neurons and examine the Inconsistency Rate (IR) of the DNN model as a result. The IR metric is defined as follows:

$$IR\,(X) = \frac{\sum_{x \in X} \mathbb{1}(\mathcal{G}(x) \neq \mathcal{L}(x))}{|X|}, \tag{8}$$

where X is a set of test inputs. $\mathcal{G}(x)$ denotes the predicted label of x after flipping the outputs of suspicious neurons. $\mathcal{L}(x)$ represents the actual label of x. $\mathbb{1}$ is the indicator function. The IR metric indicates the percentage of inputs in a set for which the prediction results differ from the actual labels after flipping suspicious neurons in a DNN model. The IR metric and accuracy are conceptually similar, as both evaluate model performance, though from different perspectives. Accuracy focuses on the number of correctly predicted samples, whereas IR counts the number of incorrectly predicted samples. We chose to adopt the IR as a metric based on its use in prior work [74]. It indicates the percentage of inputs in a set for which the prediction results differ from the actual labels after flipping suspicious neurons in a DNN model. The IR metric is complementary to accuracy. Once the IR is calculated, accuracy can be easily derived using the equation: $Accuracy = 1 - IR$. A lower IR corresponds to higher accuracy.

This paper adopts the parameter λ to control the size of suspicious neurons. It also represents the proportion of neurons with flipped weights for each model. To avoid the large accuracy degradation caused by flipping a large number of neurons in DNN models, the parameter λ is varied from 0 to 0.6 with an interval of 0.1. Given an input set, we first analyze the IR results produced by the original DNN model without neuron ablation (When $\lambda = 0$). We then compare these results with the IR values obtained after ablation of the suspicious neurons. A lower IR value for λ values ranging from 0.1 to 0.6, compared to the original IR, indicates that flipping the weights of suspicious neurons helps to mitigate certain errors in the model. In addition, the optimal parameter setting is determined by identifying the λ value that results in the lowest IR.

- **RQ2: How effective is SUNTest in generating fault-triggering test inputs?**

RQ2 investigates the fault detection capability and fault diversity triggering capability of SUNTest compared to baseline approaches. Similar to traditional software testing, a test input generation approach that can trigger more faults is more effective at revealing software defects. Therefore, we compare the number of triggered faults (*i.e.*, the number of error-revealing test inputs) in the test inputs generated by SUNTest and baseline approaches. The investigated generation approaches adopt the same test input selection strategy to select seed inputs of the same size for iteratively generating test inputs.

To uncover diverse erroneous behaviors in DNN models, we expect the test input generation approach to detect not only a greater number of faults but also a variety of fault types. We adopt the number of triggered fault types to evaluate the fault diversity triggering capability of the compared approaches. Given a test input x being misclassified by a DNN model, its fault type is defined as a tuple $(\mathcal{L}(x) \rightarrow \mathcal{G}(x))$ consisting of the actual label and the predicted label [19]. $\mathcal{L}$ denotes the actual label of x and $\mathcal{G}(x)$ denotes the predicted label, respectively. Taking the MNIST dataset as an example, if a handwritten digit image x with a true label of '3' is misclassified as '5', the fault type of x is denoted as $(3 \rightarrow 5)$. For an input set with 10 categories, the total number of distinct fault types is $10 \times 9 = 90$. Given a set of generated test inputs X, the number of triggered distinct fault types (FT) is defined as follows:

$$FT\,(X) = |\{(\mathcal{L}(x) \rightarrow \mathcal{G}(x)) \mid x \in X\}|, \tag{9}$$

where $\{(\mathcal{L}(x) \rightarrow \mathcal{G}(x)) \mid x \in X\}$ denotes the set of unique fault types triggered by X.

Coverage metrics of DNN models provide a quantifiable way to measure the diversity of a test suite. RQ2 further investigates the effectiveness of SUNTest in generating diverse test inputs by analyzing the coverage of the generated inputs. We adopt two typical coverage criteria, *i.e.*, Distance-based Surprise Coverage (DSC) [36, 37] and Neural Coverage (NLC) [78] in the evaluation. Specifically, the Distance-based Surprise Adequacy (DSA) metric quantifies the distance between the neuron output values of an input and the neuron output values of the training set. The DSC criterion quantifies the range of DSA values covered by a test suite, reflecting its diversity relative to the training set. It measures the test adequacy by assessing the coverage of the test suite over the training set. NLC is designed to capture the divergence and correlation in distributions formed by neuron outputs, demonstrating its effectiveness in assessing the diversity of a test suite.

Furthermore, RQ2 employs Inception Score (IS) [60] and Fréchet Inception Distance (FID) [28] as quantitative metrics to assess the quality of the generated test inputs. These metrics are widely recognized in both the AI and software engineering communities for assessing image quality [26, 78]. The IS metric evaluates two key attributes of the generated inputs: each input contains a clear object and exhibits high diversity. FID measures the distance between the generated and real images at the feature level.

- **RQ3: Does the test inputs generated by SUNTest guide the model retraining more effectively?**

The retraining process plays a key role in the model enhancement, facilitating the correction of learned errors and the acquisition of new knowledge. RQ3 aims to evaluate the effectiveness of SUNTest in guiding model enhancement through retraining with newly generated test inputs. Due to the potential problems of overfitting or catastrophic forgetting [8] when retraining models, we retrained DNN models using mixed training sets. For each model, newly generated test inputs produced by each test input generation approach are merged with the original training data, forming a combined training set utilized for retraining the model. The retraining process involved an additional five epochs. The hyperparameters used during retraining were set in the same way as those for the pre-trained DNN models. The retraining process was repeated 5 times to mitigate the effects of randomness, and average accuracy improvements were reported as the final results.

- **RQ4: What are the effects of SUNTest components in generating test inputs?**

The SUNTest algorithm consists of three main components: suspicious neuron localization, the hybrid fitness function, and the adaptive selection strategy for mutation operators. RQ4 aims to perform an ablation study to explore the contributions of each of these components to the overall effectiveness of the algorithm. We evaluate SUNTest and its variants by reporting the number of error-revealing test inputs, the number of triggered unique fault types, and the average accuracy improvements achieved in DNN models.

- **RQ5: How efficient is SUNTest?**

RQ5 aims to investigate the efficiency of SUNTest in localizing suspicious neurons and generating test inputs. Regarding the execution time cost for localizing suspicious neurons, we separately record the time overhead for both neuron spectrum analysis and neuron suspiciousness measurement. In terms of test input generation efficiency, we calculate the average execution time cost of four test input generation approaches across all seed selection strategies. To ensure consistency, each test input generator is applied to a fixed number of seed inputs and follows the same iteration stopping condition.

4.2 Datasets and DNN models

The experiments were conducted on four well-established datasets: MNIST [42], CIFAR10 [38], SVHN [53], and CIFAR100 [39]. MNIST is a dataset consisting of handwritten digits, with 60,000 training and 10,000 testing inputs. The CIFAR10 dataset comprises 50,000 training inputs and 10,000 testing inputs, including objects such as birds and airplanes. The SVHN dataset captures street-view house numbers from real-world scenarios. It includes 73,257 training images for training and 26,032 images for testing. CIFAR100 is a dataset consisting of 100

Table 2. Datasets and DNN models

Dataset	DNN model	# Layers	# Trainable Parameters	# Neurons	Testing Accuracy
MNIST	LeNet5	10	107,786	120	98.58%
	ConvNet_1	12	1,218,634	128	91.06%
CIFAR10	ConvNet_2	25	2,915,114	512	77.12%
	VGG16	23	2,856,010	256	84.32%
SVHN	ConvNet_3	19	6,530,250	128	91.00%
	VGG16	23	2,856,010	256	92.92%
CIFAR100	VGG19	45	15,038,116	512	68.95%
	ResNet18	69	11,486,826	512	67.12%

object classes, comprising 50,000 training images and 10,000 testing images. For each dataset, two DNN models with different architectures were employed for evaluation. For the MNIST dataset, a LeNet5 model [41] and a Convolution Neural Network (CNN) model (ConvNet_1) were trained. We used a VGG16 model [62] and a CNN model (ConvNet_2) for the CIFAR10 dataset. For SVHN, a LeNet5 model and a CNN model (ConvNet_3) were applied. For CIFAR100 dataset, we used a VGG19 model [62] and a ResNet18 model [27]. Table 2 provides detailed information on the datasets and DNN models investigated in this work. The number of neurons in the target layer for each DNN model is presented in the fifth column.

4.3 Data Simulation

Figure 3 illustrates the data simulation procedure. To mimic the testing context involving data distribution shifts, we employed four commonly used adversarial attack strategies, *i.e.*, Fast Gradient Sign Method (FGSM) [23], Basic Iterative Method (BIM-a, BIM-b) [40], and Jacobian-based Saliency Map Attack (JSMA) [56]. These strategies were implemented using the Python library Cleverhans [22].

RQ1 evaluates the inconsistency rate of DNN models on mixed input sets with varying compound ratios after flipping suspicious neurons. We conducted a merging process to integrate the original test inputs with adversarial examples. Specifically, five mixed input sets were constructed with compound ratios ranging from 0% to 80%, in 20% intervals. For instance, '80% + 20%' indicates that 80% of the inputs in the mixed set were derived from the original testing set, while the remaining 20% were sourced from the adversarial example set. A '0% + 100%' set denotes that all inputs were derived from the adversarial example set. To answer RQ2 and RQ4, we used the original testing set to simulate an unlabeled collection of test inputs for selecting seed inputs for generation. The generated test inputs, combined with the original training set, were subsequently used for model retraining. For RQ3 and RQ4, inputs in the original testing set that were not used as seeds were merged with corresponding adversarial examples to form an evaluation set. Following the data simulation process for RQ1, five mixed input sets with varying compound ratios were constructed to assess the effectiveness of the retraining process.

4.4 Baseline Approaches

4.4.1 Suspicious Neuron Localization Approaches. To answer RQ1, DeepFault [15] and NNSlicer [81] are utilized as baseline approaches.

- **DeepFault** [15] employs SBFL techniques to identify suspicious neurons with a fixed neuron activation threshold.
- **NNSlicer** [81] quantifies the contribution of each neuron and each synapse through forward and backward analyses, and calculates dynamic slices of DNN models as the neurons and synapses with larger contributions. Despite its primary focus not being the identification of suspicious neurons, we regard

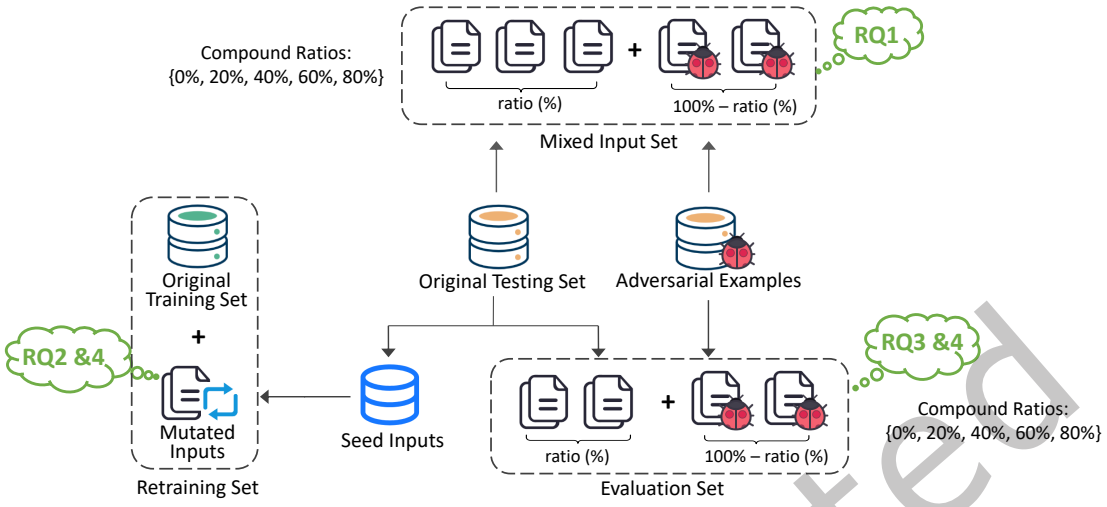

Fig. 3. The Procedure of Data Simulation

NNSlicer as a baseline approach. This is attributed to its capability to model the average behavior of each neuron by leveraging the average output values obtained from the entire training set.

RQ1 investigates the impact of different neuron activation thresholds on the localization of suspicious neurons, including *i.e.*, DeepFault's fixed activation threshold, NNSlicer's average neuron outputs, and SUNTest's dynamic activation thresholds. For each approach, the average IR results of three SBFL formulas are recorded.

4.4.2 Test Input Generation Approaches. To evaluate the effectiveness of SUNTest in generating test inputs, three state-of-the-art approaches adhering to the MGF framework are employed as baseline approaches. DLFuzz [25] and ADAPT [43] focus on covering specific sets of neurons. RobOT [68] adopts model robustness-correlated testing metrics as guidance for generating test inputs.

- **DLFuzz** [25] generates test inputs aimed at maximizing neuron coverage as defined by Pei et al. [58] and the prediction difference between the original input and the mutated input. It proposes four neuron selection heuristics to prioritize neurons covered frequently, rarely, neurons with top weights, and neurons near the activation threshold.
- **ADAPT** [43] adopts an adaptive neuron selection strategy by considering both static and dynamic neuron features. This strategy is employed to generate test inputs under the guidance of the multi-granularity neuron coverage metrics defined by Ma et al. [49].
- **RobOT** [68] defines model robustness metrics, *i.e.*, Zero-Order Loss (ZOL) and First-Order Loss (FOL), to direct the fuzzing process for generating test inputs.

4.4.3 Variants of SUNTest. To explore the effects of different components of SUNTest, we introduce four variants as follows:

- **SUNTest-RO** replaces the adaptive selection strategy of mutation operators in SUNTest with random selection. The abbreviation 'RO' denotes 'Random Operator'. This variant aims to investigate whether integrating historical execution information of mutation operators could induce diverse misbehaviors.

- **SUNTest-RN** replaces the suspicious neurons adopted in the fitness function $f(x)$ with randomly selected neurons. The abbreviation 'RN' stands for 'Random Neuron'. By randomizing the selection of neurons, this variant is designed to assess the impact of suspicious neurons on the overall performance.
- **SUNTest-ER** substitutes the hybrid fitness function in SUNTest with the fitness function $f(x)$, focusing solely on the error-revealing capability feedback of test inputs. The abbreviation 'ER' signifies 'Error-Revealing'. The comparison between SUNTest-ER and SUNTest aims to investigate the usefulness of the input diversity guidance in SUNTest.
- **SUNTest-ID** replaces the hybrid fitness function in SUNTest with the fitness function $g(x)$, which solely considers the diversity feedback of test inputs. The abbreviation 'ID' denotes 'Input Diversity'. This variant is designed to investigate the usefulness of the error-revealing capability guidance in SUNTest.

4.4.4 Test Input Selection Approaches. This paper constructs the initial seed list for generating test inputs by employing various test input selection approaches. This process facilitates the construction of a seed list containing test inputs that are more likely to trigger misclassifications. For each dataset under investigation, six selection approaches are performed on the correctly classified inputs extracted from the original testing set. KM-ST [68] and BE-ST [68] are designed based on the calculation of FOL values of test inputs. FOL measures the extent to which the highest loss could be obtained within the seed's neighborhood. Both DeepGini [18] and MCP [61] are uncertainty-based selection metrics, leveraging output probabilities of DNN models.

- **Random** is the basic and simplest selection approach, where each input has the same probability of being selected.
- **KM-ST** [68] divides the range of FOL values of test inputs into k sections, and randomly selects the same number of test inputs from each range.
- **BE-ST** [68] equally combines test inputs with small and large FOL values.
- **DeepGini** [18] utilizes the output probabilities of DNN models to measure the Gini impurity of the test inputs. It prioritizes test inputs with the most uncertainty, enabling the detection of potentially misclassified test inputs.
- **MCP** [61] clusters test inputs into different decision boundary areas by calculating the predicted probabilities for the top two classes. Test inputs with high priorities (*i.e.*, the ratio of the probability of the first class to that of the second class) are uniformly selected from each boundary area.
- **MOON** [24] is our previous research that addresses the test input selection problem by formulating it as a search-based testing problem. By tailoring a multi-objective optimization algorithm, it directs the search process to maximize the activation of suspicious neurons while promoting diversity in neuron behaviors.

4.5 Hyperparameter Configurations

SUNTest determines the activation threshold for each neuron based on its output distributions. The parameter b represents the number of equal intervals into which the output range is divided. Similarly, DeepGuage [48] partitions the output range of each neuron into several equal sections. Following DeepGuage's configuration, b is set to 1000. The parameter K denotes the number of the most frequent intervals used to characterize high-frequency regions of neuron outputs. If K is set large, it may introduce certain output intervals with low frequency. Thus, it is set to 100. Both parameters λ and q are adopted to control the size of suspicious neurons. The parameter λ controls the size of localized suspicious neurons. It also represents the proportion of neurons with flipped outputs for each model. The identified suspicious neurons not only contribute to errors but may also influence the classification of other inputs. To avoid the large accuracy degradation caused by flipping a large number of neurons in DNN models, the parameter λ is varied from 0.1 to 0.6 with an interval of 0.1 in the experiment. To obtain the error-revealing capability feedback guided by suspicious neurons, the parameter q is utilized to further

select neurons from the set of identified suspicious neurons. This parameter controls the size of dynamically combined suspicious neurons used for test input generation. If q is set too small, fewer suspicious neurons will be used to measure the error-revealing capability of test inputs. Therefore, it is set to 0.8.

To compare SUNTest with existing MGF approaches under uniform conditions, test inputs are iteratively generated with a fixed number of seed inputs until the maximum iteration limit is reached. The number of seed inputs is set to 200. Assigning the appropriate number of iterations is crucial. A large number results in significant differences in input semantics, while a small number fails to produce noticeable changes in the seed inputs. Therefore, the maximum number of iterations is set to 5. Additionally, regarding the test input selection strategies, for both BE and KM, the range of FOL is divided into 20 sections. In the case of MOON, the population size is set to 100, and the algorithm undergoes 50 iterations of evolution. For the coverage criteria DSC utilized in the experiment, the number of buckets is set to 1,000. The upper bound of SC values for test inputs is set to 10, while the lower bound is set to 0.

4.6 Statistical Analysis Methods

To determine the statistical significance of SUNTest's performance compared to baseline approaches, we employ two statistical analysis methods: the Wilcoxon signed-rank test [7] and Cliff's delta $|\delta|$ [59] to quantify the differences in results. The confidence level for the Wilcoxon signed-rank test is set at 0.05. A p-value below 0.05 indicates that SUNTest demonstrates a statistically significant advantage over other baseline approaches. Conversely, when the p-value is greater than 0.05, there is no considerable difference observed between the two compared approaches. For Cliff's delta, the magnitude of differences is assessed using the following thresholds: $|\delta| < 0.147$ as negligible, $0.147 < |\delta| < 0.330$ as small, $0.330 < |\delta| < 0.474$ as medium, and $|\delta| \geq 0.474$ as large. In experiments, SUNTest is considered to significantly outperform baseline approaches when the p-value is less than 0.05 and Cliff's delta $|\delta|$ is greater than 0.147.

5 EXPERIMENT RESULTS

5.1 Answer to RQ1: Effectiveness in Suspicious Neuron Localization

Table 3 presents the inconsistency rates observed in DNN models after flipping the weights of suspicious neurons. The 'Ratio' column illustrates different combinations of original test inputs and adversarial examples within mixed input sets (*i.e.*, mimicking various data distributions). When λ is set to 0, no modifications (no neurons were flipped) were applied to the DNN models. In this case, the reported inconsistency rates serve as baselines. The decreases in IR values, when compared to the original values ($\lambda = 0$), demonstrate the effectiveness of flipping suspicious neurons in rectifying erroneous neuron behaviors.

As presented in Table 3, the inconsistency rates of DNN models can be reduced by reversing the weights of suspicious neurons, particularly when λ is set to 0.1 or 0.2. In these cases, the IR results obtained by the three approaches are mostly lower than the original value. As highlighted by the underlined results in Table 3, a λ value of 0.1 results in the highest frequency of the lowest IR values. As the λ value increases, the IR values naturally rise, eventually exceeding the original value. This can be attributed to the fact that the localized suspicious neurons not only contribute to the errors but also potentially play a role in the classification of other inputs. Additionally, it is obvious that as the proportion of adversarial examples in the mixed dataset increases, the inconsistency rate of DNN models also increases. The reduction in IR results achieved by SUNTest becomes more pronounced as the proportion of adversarial examples rises. For example, in the CIFAR100-VGG model, the original inconsistency rate is 44.26%. SUNTest achieves the lowest inconsistency rates of 42.67%, 54.94%, 67.27%, 77.63%, and 90.83% across five compound ratios, corresponding to decreases of 1.59%, 2.32%, 3.47%, 6.15%, and 6.23%, respectively.

Based on the observations above, the test input generation algorithm sets λ to 0.1. In the following part, we focus primarily on the results of achieved inconsistency rates when λ is set to 0.1 and 0.2. As shown in Table 4, the

Table 3. The Inconsistency Rates (%) of DNN Models after Flipping Suspicious Neurons

Dataset	Ratio	Approach	The Proportion of Selected Suspicious Neurons (λ)													
			0	0.1	0.2	0.3	0.4	0.5	0.6	0	0.1	0.2	0.3	0.4	0.5	0.6
			ConvNet							LeNet5						
MNIST	80%+20%	DeepFault	27.03	25.57	27.85	35.65	59.09	81.38	93.24	21.08	20.16	26.93	30.67	37.80	66.15	71.59
		NNSlicer		25.90	28.86	37.76	55.11	74.24	**85.24**		24.56	32.94	37.59	43.38	56.18	**70.65**
		SUNTest		**25.38**	**26.75**	**32.95**	**51.75**	**72.49**	87.76		**19.71**	**25.43**	**28.71**	**36.46**	**55.19**	72.57
	60%+40%	DeepFault	45.33	41.53	42.87	49.09	67.26	84.62	93.23	40.70	38.40	41.08	43.63	51.16	71.51	75.52
		NNSlicer		41.91	44.15	50.79	64.03	78.34	**86.70**		40.02	45.66	49.45	53.58	63.40	**74.80**
		SUNTest		**40.69**	**42.20**	**46.80**	**61.17**	**77.54**	88.59		**36.39**	**39.18**	**42.11**	**43.60**	**62.70**	75.91
	40%+60%	DeepFault	63.49	57.39	57.81	62.30	75.38	87.57	93.20	60.46	**52.65**	55.20	56.74	57.64	77.12	79.89
		NNSlicer		57.96	59.47	63.89	72.90	82.27	**88.04**		55.04	58.11	61.19	63.63	**70.66**	79.35
		SUNTest		**56.39**	**56.81**	**60.81**	**70.80**	**80.59**	89.49		54.25	**53.03**	**55.65**	**56.84**	73.89	**78.65**
	20%+80%	DeepFault	81.86	74.25	72.69	75.54	83.29	90.63	93.06	80.11	70.96	69.11	69.77	70.66	82.80	84.43
		NNSlicer		75.07	74.69	76.80	81.60	86.16	**89.28**		**70.47**	71.00	73.10	73.90	**78.27**	84.01
		SUNTest		**73.14**	**72.16**	**74.51**	**79.30**	**84.65**	90.15		71.61	**67.06**	**69.15**	**70.22**	80.05	**82.26**
	0%+100%	DeepFault	100.0	88.97	87.55	88.62	91.28	93.75	93.11	99.80	**85.23**	83.26	83.90	83.64	**88.29**	88.71
		NNSlicer		89.76	89.85	89.78	90.45	90.17	**90.73**		85.72	84.02	84.94	83.94	84.88	87.47
		SUNTest		**87.17**	**86.88**	**87.33**	**88.17**	**87.74**	91.79		85.33	**81.02**	**82.76**	**83.57**	86.02	**86.80**
			ConvNet							VGG						
CIFAR10	80%+20%	DeepFault	37.78	38.02	38.53	39.47	42.75	53.77	63.66	32.16	32.22	34.36	47.21	75.51	92.69	95.74
		NNSlicer		**37.26**	37.00	37.49	38.63	44.40	65.08		32.17	34.58	47.69	77.10	92.20	95.60
		SUNTest		37.33	**36.82**	**37.33**	**37.72**	**38.41**	**41.05**		**32.07**	**33.70**	**46.16**	**73.32**	**91.03**	**95.08**
	60%+40%	DeepFault	53.12	52.79	52.91	53.39	55.73	63.98	70.52	48.42	47.89	48.96	58.86	78.81	91.12	93.07
		NNSlicer		52.33	51.88	51.87	52.55	56.63	71.93		47.85	49.19	58.97	80.07	90.90	92.91
		SUNTest		**51.74**	**50.69**	**51.77**	**52.02**	**52.35**	**53.94**		**46.89**	**48.65**	**57.28**	**76.92**	**89.80**	**92.23**
	40%+60%	DeepFault	68.18	67.51	67.25	67.35	68.77	74.24	78.32	65.0	64.15	64.40	69.93	82.12	89.79	90.38
		NNSlicer		67.35	66.58	67.30	66.43	68.96	78.65		64.26	64.39	70.53	93.20	89.83	**90.26**
		SUNTest		**67.10**	**66.01**	**66.33**	**66.21**	**66.34**	**66.23**		**63.88**	**63.02**	**68.25**	**81.03**	**88.96**	90.74
	20%+80%	DeepFault	83.04	81.86	81.25	81.02	81.58	84.25	85.95	81.20	80.01	79.65	81.89	85.78	88.28	87.86
		NNSlicer		82.04	81.27	80.45	80.15	81.15	85.54		79.93	79.46	82.76	86.48	88.73	87.75
		SUNTest		**80.59**	**79.77**	**79.54**	**79.67**	**80.22**	**82.74**		**78.17**	**78.09**	**80.29**	**85.16**	**87.17**	**87.32**
	0%+100%	DeepFault	98.11	96.44	95.50	94.89	94.55	94.51	93.71	97.66	95.92	94.73	93.51	89.31	**86.95**	85.22
		NNSlicer		96.87	96.73	94.65	93.90	93.31	92.27		95.84	94.30	93.25	89.69	87.64	**85.08**
		SUNTest		**95.76**	**93.80**	**92.83**	**91.98**	**92.17**	**91.88**		**94.67**	**93.19**	**92.03**	**88.72**	87.13	85.11
			ConvNet							VGG						
SVHN	80%+20%	DeepFault	27.18	27.31	27.90	29.13	38.37	60.22	75.04	25.08	26.05	34.68	57.60	81.95	92.84	95.64
		NNSlicer		27.21	27.83	29.25	37.45	57.03	73.62		26.44	37.36	58.20	80.45	91.68	95.37
		SUNTest		**27.05**	**27.25**	**28.11**	**33.46**	**54.23**	**73.58**		**25.80**	**34.01**	**56.29**	**78.17**	**90.33**	**95.24**
	60%+40%	DeepFault	45.29	44.89	44.87	45.17	50.80	65.33	77.46	42.99	43.72	51.54	66.94	83.21	91.40	93.75
		NNSlicer		44.95	44.84	44.96	51.07	66.79	77.61		43.43	49.58	66.68	84.32	92.16	94.01
		SUNTest		**43.98**	**43.67**	**44.21**	**48.51**	**63.58**	**77.27**		**42.76**	**44.46**	**65.02**	**82.04**	**90.13**	**93.06**
	40%+60%	DeepFault	63.61	62.70	62.12	61.27	64.34	73.46	81.18	61.03	60.88	64.56	75.72	86.80	91.47	92.38
		NNSlicer		62.72	61.95	60.92	63.90	73.49	**80.19**		61.06	65.87	75.76	86.05	91.13	92.21
		SUNTest		**62.16**	**61.81**	**59.36**	**63.17**	**72.31**	81.90		**59.27**	**62.07**	**74.39**	**85.34**	**90.28**	**91.03**
	20%+80%	DeepFault	81.67	80.27	79.12	77.14	77.80	81.91	84.96	78.91	78.23	79.52	84.73	89.34	90.88	90.15
		NNSlicer		80.27	78.84	76.66	**76.54**	80.28	**82.97**		78.30	80.20	84.59	89.01	90.85	90.70
		SUNTest		**79.64**	**78.34**	**76.02**	77.80	**79.13**	84.08		**77.27**	**78.36**	**84.17**	**88.72**	**90.13**	**90.45**
	0%+100%	DeepFault	99.86	98.02	96.32	93.25	91.36	90.29	88.93	96.85	95.52	94.31	93.73	91.83	90.23	89.18
		NNSlicer		97.98	**95.92**	92.67	**89.34**	**86.96**	**85.55**		95.49	94.27	93.28	91.88	90.55	89.12
		SUNTest		**97.02**	96.88	**92.07**	92.06	90.01	86.30		**94.57**	**93.73**	**93.46**	**91.02**	**90.13**	**88.14**
			VGG							ResNet						
CIFAR100	80%+20%	DeepFault	44.26	43.96	51.60	77.81	95.02	99.09	99.74	45.69	48.87	59.42	77.91	93.12	98.52	99.72
		NNSlicer		43.59	45.82	**58.24**	**80.78**	**95.68**	**98.91**		49.05	58.42	75.68	**91.88**	**98.34**	**99.62**
		SUNTest		**42.67**	**45.20**	58.59	81.45	96.58	99.09		**47.21**	**57.27**	**76.08**	92.33	98.37	99.68
	60%+40%	DeepFault	57.26	56.03	61.61	81.98	95.65	99.05	99.60	58.78	60.02	67.90	82.29	94.44	98.76	99.68
		NNSlicer		55.78	57.33	**66.61**	**84.26**	**96.25**	**98.87**		60.01	67.08	80.64	**93.33**	**98.58**	**99.60**
		SUNTest		**54.94**	**56.48**	66.82	84.91	96.93	99.04		**57.89**	**65.78**	**79.96**	93.81	98.63	99.64
	40%+60%	DeepFault	70.74	68.80	72.21	86.57	96.59	99.15	99.55	71.29	70.86	76.45	86.90	95.73	98.85	99.67
		NNSlicer		68.74	69.35	**75.48**	**88.18**	**96.94**	**99.04**		70.98	75.94	85.59	**94.85**	**98.75**	**99.61**
		SUNTest		**67.27**	**68.30**	75.75	88.67	97.46	99.11		**69.12**	**73.90**	**84.43**	95.22	98.80	99.62
	20%+80%	DeepFault	83.78	81.02	82.53	90.85	97.38	99.15	99.49	84.12	81.65	84.81	91.24	96.94	99.06	99.67
		NNSlicer		81.13	80.94	84.32	**91.65**	**97.47**	**98.99**		81.63	84.28	90.43	**96.44**	**99.01**	**99.61**
		SUNTest		**77.63**	**79.17**	**83.80**	92.07	97.94	99.05		**80.13**	**82.87**	**90.18**	96.62	99.03	99.62
	0%+100%	DeepFault	97.06	93.59	92.92	95.21	98.22	99.20	99.39	96.88	92.46	93.29	95.58	98.15	99.20	99.65
		NNSlicer		93.94	93.07	93.17	95.53	**98.18**	**99.08**		92.37	93.02	95.54	**98.09**	**99.23**	**99.61**
		SUNTest		**91.67**	**90.83**	**92.93**	**95.01**	98.48	99.13		**90.26**	**91.60**	**94.43**	98.16	99.25	99.63

For each λ configuration (ranging from 0.1 to 0.6), the lowest inconsistency rate among the three localization approaches is highlighted in bold. The lowest inconsistency rate, achieved by each approach across λ configurations ranging from 0.1 to 0.6, is underlined.

Table 4. Comparison Results of Inconsistency Rates Achieved by Different Approaches

Dataset	Model	λ Configuration	# Top-1 (Formula)			# Top-1 (Approach)			Avg Dec		
			Tarantula	Ochiai	DStar	DeepFault	NNSlicer	SUNTest	w.r.t Orig	w.r.t DeepFault	w.r.t NNSlicer
MNIST	LeNet	0.1	5	0	0	2	1	2	6.97	0.02	1.70
		0.2	5	0	0	0	0	5	7.29	1.97	5.20
	ConvNet	0.1	5	0	0	0	0	5	6.99	0.99	1.57
		0.2	0	2	3	0	0	5	6.58	0.79	2.44
CIFAR10	ConvNet	0.1	0	5	0	0	1	4	1.54	0.82	0.67
		0.2	3	0	2	0	0	5	2.63	1.67	1.27
	VGG	0.1	5	0	0	0	0	5	1.75	0.90	0.87
		0.2	0	0	5	0	0	5	1.56	1.09	1.05
SVHN	ConvNet	0.1	0	5	0	0	0	5	1.55	0.67	0.66
		0.2	0	4	1	0	1	4	1.93	0.48	0.29
	VGG	0.1	0	5	0	0	0	5	1.04	0.95	1.01
		0.2	0	4	1	0	0	5	-1.55	2.40	2.93
CIFAR100	VGG	0.1	0	1	4	0	0	5	3.78	1.84	1.80
		0.2	0	3	2	0	0	5	2.62	4.18	1.31
	ResNet	0.1	0	5	0	0	0	5	2.43	1.85	1.89
		0.2	0	1	4	0	0	5	-2.93	2.09	1.46
	Total	0.1	15	21	4	2	2	36	2.90	0.89	1.13
		0.2	8	14	18	0	1	39	2.01	1.63	1.77

'# Top-1 (Formula)' columns report the number of instances where a suspiciousness measure (Tarantula, Ochiai, or DSatr) outperforms the others across five data compound ratios. The '# Top-1 (Approach)' columns provide the frequency with which each approach (DeepFault, NNSlicer, or SUNTest) achieves the lowest inconsistency rate across five data compound ratios. Since we have a total of 8 models and 5 data compound ratios, there are 40 result instances in total (8×5=40). The '# Top-1 (Formula)' and '# Top-1 (Approach)' columns display, in the 'Total' rows, the total number of instances where the top-1 rank is achieved. As observed in the '# Top-1 (Formula)' results, different SBFL formulas exhibit varying performance across different datasets and models. When λ is set to 0.1, the Tarantula formula outperforms the others for the MNIST and CIFAR10 datasets, while Ochiai is effective for the SVHN and CIFAR100 datasets. Regarding the 'Total' results, Ochiai outperforms both Tarantula and DStar, achieving the lowest inconsistency rate 21 times. Tarantula and DStar follow, achieving the top rank 15 times and 4 times, respectively. When λ is set to 0.2, DStar outperforms Tarantula and Ochiai, achieving the top rank 18 times. Based on these observations, for the test input generation algorithm, we employ Tarantula for MNIST and CIFAR10, and Ochiai for SVHN and CIFAR100. Compared with baseline approaches, SUNTest achieves the lowest inconsistency rates across almost all instances. It obtains the top rank 36 times when λ=0.1 and 39 times when λ=0.2, whereas DeepFault achieves the top rank 2 times in both configurations, and NNSlicer only achieves it once.

For each investigated model, the 'Avg Dec' columns show the average decreases in inconsistency rates across five data compound ratios, achieved by SUNTest with respect to the original values (λ is configured with 0), DeepFault, and NNSlicer, respectively. The 'Avg Dec' columns in the 'Total' rows display the average decreases in inconsistency rates across all investigated models. As indicated by the 'Dec (w.r.t Org)' results, SUNTest achieves inconsistency rate decreases of up to 7.29%, 6.99%, 2.63%, 1.75%, 1.93%, 1.04%, 3.78%, and 2.43% for the investigated models, respectively. On average, compared to the original values, SUNTest reduces the inconsistency rates by 2.90% and 2.01% across the investigated models in the 0.1 and 0.2 λ configurations. Additionally, compared to DeepFault, SUNTest achieves average reductions in inconsistency rates of up to 1.97%, 0.99%, 1.67%, 1.09%, 0.67%, 2.40%, 4.18%, and 2.09% across the investigated DNN models, respectively. Compared to NNSlicer, SUNTest

achieves average reductions of up to 5.20%, 2.44%, 1.27%, 1.05%, 0.66%, 2.93%, 1.80%, and 1.89%, respectively. These results indicate that SUNTest is effective in localizing suspicious neurons.

> Answer to RQ1: SUNTest outperforms baseline approaches in localizing suspicious neurons by employing dynamic neuron activation thresholds based on neuron output distributions. Specifically, after flipping suspicious neurons under the λ configurations of 0.1 and 0.2, SUNTest achieves reductions in inconsistency rates of up to 4.18% and 5.20%, respectively, compared to DeepFault and NNSlicer.

5.2 Answer to RQ2: Effectiveness in Test Input Generation

SUNTest is compared with baseline approaches in terms of the effectiveness of test input generation from four perspectives, *i.e.*, the capability of fault detection, the capability of triggering diverse fault types, the coverage achieved by the newly generated test inputs, and the validity of the generated inputs. Table 5 presents the average number of error-revealing test inputs and triggered fault types for each approach adopting different seed selection strategies. The '# Inps' rows report the number of generated test inputs that trigger misclassifications, while the '# Typs' rows show the number of distinct fault types. The '↑' rows detail the average increases (A-B) in results (*i.e.*, average '# Inps' and '# Typs' results achieved across investigated models) obtained by SUNTest (A) compared to the baseline approaches (B) when using the same seed selection strategy. The '↑ (%)' rows present the average improvement rates (*i.e.*, (A-B)/B) of the results. Table 6 details the average DSC and NLC results achieved by the generated test inputs. In Tables 5 and 6, the 'Avg' columns present the average results across all 6 seed selection strategies for each test input generation approach. Since we compare SUNTest with the other 3 baseline approaches with 6 seed selection strategies, the total number of combinations of the test input generation approaches is $4 \times 6 = 24$. The 'Avg Rank (I)' rows represent the average rank of each approach when adopting the same seed selection strategy. The 'Avg Rank (II)' rows denote the average rank of each combination. The lower the average rank, the better the approach. The '# Top-1 (I)' rows present the frequency with which each approach is ranked as the top among the four, given the same seed selection strategy. The '# Top-1 (II)' rows detail the number of times each combination achieves the first rank.

5.2.1 Fault Detection. In terms of the average number of error-revealing test inputs across all seed selection strategies, as indicated by 'Avg' results, SUNTest outperforms baseline MGF approaches in nearly all investigated models, with the exception of CIFAR10-ConvNet. Based on the '↑' results for average '# Inps' obtained from 6 seed selection strategies, we observe that SUNTest is able to find more error-revealing test inputs compared to baseline approaches, with average increases ranging from 100.3 to 310.5. Considering the '↑ (%)' results, SUNTest obtains average improvement rates of 17.7%, 162.9%, and 37.5% compared to DLFuzz, ADAPT, and RobOT, respectively. In the view of the '# Top-1 (I)' results, SUNTest achieves the top-1 best average performance 7 times, whereas the second-best, DLFuzz, only achieves it once. Additionally, SUNTest achieves the best average rank of 1.25, while the average ranks for DLFuzz, ADAPT, and RobOT are 2.50, 3.88, and 2.38, respectively. In general, SUNTest achieves the best results in most investigated DNN models, indicating its capability to detect more faults.

When adopting Random, BE, KM, MCP, and MOON strategies, SUNTest significantly outperforms DLFuzz, ADAPT, and RobOT across almost all investigated models. For example, with the Random, BE, KM, MCP, and MOON strategies, SUNTest achieves the lowest average ranks among the four approaches, with values of 1.25, 1.38, 1.25, 1.38, and 1.38, respectively. Nevertheless, when employing the DeepGini strategy, SUNTest performs worse than baseline approaches. Specifically, with the DeepGini strategy, DLFuzz, ADAPT, and RobOT achieve average ranks of 1.63, 2.88, and 2.13, respectively, whereas SUNTest obtains an average rank of 3.38. Among all 24 combinations, DLFuzz paired with DeepGini yields the lowest average rank of 1.50, as indicated by the 'Avg Rank (II) results. It achieves the highest frequency of being the top-ranked combination, as shown by the '# Top-1 (II)

Table 5. Comparison Results in Generating Error-Revealing Test Inputs and Triggering Fault Types

Dataset	Model	Metric	DLFuzz							ADAPT							RobOT							SUNTest						
			Random	BE	KM	DeepGini	MCP	MOON	Avg	Random	BE	KM	DeepGini	MCP	MOON	Avg	Random	BE	KM	DeepGini	MCP	MOON	Avg	Random	BE	KM	DeepGini	MCP	MOON	Avg
MNIST	LeNet5	# Inps	455.2	527.2	345.4	983.8	490.7	504.0	551.1	56.0	351.9	33.6	582.9	79.1	77.7	196.9	80.1	499.8	63.2	966.7	120.9	117.3	308.0	594.6	667.6	620.3	682.5	613.9	618.3	**632.9**
		# Typs	54.2	48.2	45.1	64.5	58.3	56.1	54.4	20.1	42.2	12.4	54.2	24.4	25.3	29.8	17.0	41.9	14.1	55.0	19.6	23.2	28.5	74.7	70.7	68.4	75.0	74.8	73.6	**72.9**
	ConvNet	# Inps	393.8	492.3	255.3	935.0	459.6	496.0	505.3	12.3	169.9	12.4	174.1	21.6	27.5	69.6	76.2	438.3	52.4	618.4	111.1	150.1	241.1	607.8	620.3	606.9	661.8	635.7	633.7	**627.7**
		# Typs	51.5	53.0	35.3	71.0	53.2	62.2	54.3	4.3	33.5	3.6	35.6	7.1	8.9	15.5	18.8	47.1	15.9	62.3	28.0	30.7	33.8	82.9	77.6	72.6	82.7	85.7	86.7	**81.4**
SVHN	ConvNet	# Inps	455.7	464.5	297.3	691.0	475.7	486.2	478.4	402.5	507.2	321.3	756.3	463.0	491.0	490.2	624.9	497.9	418.5	742.3	678.4	678.6	606.8	638.5	698.9	644.8	684.4	628.1	634.7	**655.4**
		# Typs	59.0	60.0	47.5	62.4	62.5	68	59.9	60.2	60.2	48.9	72.5	61.2	65.0	61.3	59.1	45.4	38.0	56.4	60.5	61.0	53.4	75.6	76.9	68.6	79.1	79.5	80.4	**76.7**
	VGG	# Inps	527.3	744.0	536.3	987.0	513.3	593.8	650.3	261.1	666.1	266.2	966.6	273.3	314.0	457.9	443.3	713.7	549.6	965.5	423.9	457.6	592.3	679.0	693.9	673.0	736.4	682.3	694.6	**693.2**
		# Typs	68.3	77.5	67.3	84.5	69.3	77.8	73.4	46.7	65.9	45.6	76.7	51.8	52.9	56.6	52.3	52.7	49.2	55.5	51.9	52.8	52.4	82.6	81.4	78.9	86.4	85.3	86.2	**83.5**
CIFAR10	ConvNet	# Inps	954.0	901.0	895.8	987.8	964.3	959.9	**943.8**	416.6	492.4	289.3	882.0	493.0	481.2	509.1	896.3	777.9	807.8	956.5	908.2	890.4	872.9	824.6	817.4	802.6	903.7	825.6	829.3	833.9
		# Typs	74.5	71.4	62.0	73.8	72.0	74.5	**71.4**	59.4	60.5	42.6	75.2	63.3	64.6	60.9	66.4	57.4	55.4	65.9	70.9	66.6	63.8	72.5	68.4	65.7	72.3	72.9	73.2	70.8
	VGG	# Inps	505.2	546.5	398.5	985.7	581.8	613.6	605.2	357.8	513.0	254.7	967.1	420.0	413.2	487.6	789.3	777.2	694.9	958.6	811.3	742.4	795.6	829.7	820.1	847.2	907.4	894.6	856.8	**859.3**
		# Typs	69.2	68.0	56.8	79.8	72.3	76.2	70.4	57.6	60.1	53.3	73.4	60.7	65.4	61.7	71.1	46.9	60.8	37.3	67.9	69.8	59.0	72.5	71.8	72.6	76.2	74.8	81.2	**74.9**
CIFAR100	VGG	# Inps	681.0	693.3	789.3	990.3	754	808.3	786.1	545.7	631.7	554.7	979.7	666.5	641.3	669.9	867.8	836.4	870.4	995.0	922.4	896.6	898.1	915.6	921.5	918.1	966.3	938.9	932.2	932.1
		# Typs	277.7	302.3	332.7	403	316.7	417.0	341.6	229.0	245.5	225	351.3	264.6	284.8	266.7	200.9	140.2	202.8	187.9	191	206.4	188.2	462.6	444.9	461.2	476.7	474.5	479.8	464.0
	ResNet	# Inps	774.3	813.7	825.3	990.7	824.7	849.2	846.3	674.5	787.3	778.6	989.7	763.8	823.6	802.9	882.6	899.2	919.8	995.9	911.6	894.4	917.3	916.6	926	934.2	968.9	928.7	930.6	934.2
		# Typs	326.7	370.0	367.7	412.3	373.0	414.2	377.3	270.5	295.7	299.6	319.9	292.4	326.6	300.8	239.8	182.4	224.3	157.8	215.2	241.4	210.2	406.2	391.6	410.5	424.3	414.0	427.5	412.4
↑		# Inps	157.5	122.9	213.0	-130.0	135.5	102.7	100.3	410.0	255.8	442.0	26.6	370.9	357.9	310.5	168.2	90.7	208.8	-85.9	157.5	163.2	117.1	-	-	-	-	-	-	-
		# Typs	43.6	29.1	35.5	15.2	35.5	17.8	29.2	72.7	52.5	70.9	39.2	67.0	61.9	60.4	75.5	83.7	79.8	86.8	82.1	79.3	80.9	-	-	-	-	-	-	-
↑ (%)		# Inps	32.2	23.0	61.4	-13.4	25.6	18.7	17.7	794.4	73.3	911.2	32.1	493.4	411.1	162.9	175.2	17.1	255.6	-8.8	117.3	101.4	37.5	-	-	-	-	-	-	-
		# Typs	30.1	22.6	37.9	9.6	25.7	15.3	22.0	300.3	49.6	339.3	32.6	195.9	161.1	98.8	122.1	82.7	141.4	75.2	104.7	90.8	84.9	-	-	-	-	-	-	-
Avg Rank		# Inps	2.38	2.38	2.63	1.63	2.38	2.50	2.50	4.00	3.75	3.88	2.88	4.00	3.88	3.88	2.38	2.50	2.25	2.13	2.25	2.25	2.38	1.25	1.38	1.25	3.38	1.38	1.38	1.25
(I)		# Typs	2.13	2.00	2.25	2.00	2.00	1.88	2.00	3.38	3.00	3.50	2.88	3.50	3.38	3.13	3.38	3.88	3.25	3.75	3.50	3.63	3.75	1.13	1.13	1.00	1.38	1.00	1.13	1.13
Avg Rank		# Inps	16.1	13.3	15.8	1.5	13.8	12.1	-	23.5	17.1	23.3	5.9	20.8	20.0	-	14.6	12.6	16.3	2.9	12.0	13.4	-	9.6	7.6	8.9	4.3	7.6	7.3	-
(II)		# Typs	11.9	12.0	15.6	6.0	9.4	6.3	-	19.8	16.4	21.5	9.0	17.6	15.3	-	18.0	20.3	21.3	17.4	17.9	16.1	-	5.4	7.0	7.6	3.1	3.4	2.0	-
# Top-1		# Inps	1	2	1	5	1	1	1	0	0	0	1	0	0	0	0	0	0	2	1	1	0	7	6	7	0	6	6	7
(I)		# Typs	1	1	0	1	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	7	7	8	5	8	7	7
# Top-1		# Inps	0	0	0	5	0	0	-	0	0	0	1	0	0	-	0	0	0	2	0	0	-	0	0	0	0	0	0	-
(II)		# Typs	0	0	0	0	0	0	-	0	0	0	1	0	0	-	0	0	0	0	0	0	-	0	0	0	2	0	5	-

[1] For each test input generation approach, the best result across six seed selection strategies is underlined. The best average results 'Avg' among the four test input generation approaches are highlighted in bold and displayed with a yellow-colored background.

[2] Uncolored cells signify instances where SUNTest significantly outperforms baselines under the same seed selection strategy. Cells with a gray background denote cases where SUNTest performs worse than the baseline or where there is no statistically significant difference between SUNTest and the baseline.

results. Besides, for the four test input generation approaches, DeepGini demonstrates superior performance compared to the other seed selection strategies, as indicated by the underlined results. These observations can be attributed to the fact that DeepGini tends to select the most uncertain test inputs as seeds, which are more likely to result in the mutated inputs triggering misclassifications.

5.2.2 Fault Diversity. Similar to the results of fault detection, with the exception of CIFAR10-ConvNet, SUNTest demonstrates significant superiority over DLFuzz, ADAPT, and RobOT, achieving the highest average number of triggered fault types. As indicated by the '↑' results, SUNTest detects more diverse fault types compared to baseline approaches, with increases of 29.2, 60.4, and 80.9 on average. In terms of the '↑ (%)' results, SUNTest obtains average improvement rates of 22.0%, 98.8%, and 84.9% compared to DLFuzz, ADAPT, and RobOT, respectively. For the '# Top-1 (I)' results, SUNTest achieves the top-1 best average performance 7 times, whereas the second-best, DLFuzz achieves it once. Additionally, SUNTest achieves the best average rank of 1.13, compared to the average ranks of 2.00, 3.13, and 3.75 obtained by DLFuzz, ADAPT, and RobOT, respectively. When adopting the same seed strategy, SUNTest demonstrates the lowest average rank among the four approaches. Specifically, SUNTest achieves average ranks of 1.13, 1.13, 1.0, 1.38, 1.0, and 1.13 for the 6 seed selection strategies, respectively. These

Table 6. Coverage Achieved by the Generated Test Inputs

Dataset	Model	Criterion	DLFuzz							ADAPT							RobOT							SUNTest						
			Random	BE	KM	DeepGini	MCP	MOON	Avg	Random	BE	KM	DeepGini	MCP	MOON	Avg	Random	BE	KM	DeepGini	MCP	MOON	Avg	Random	BE	KM	DeepGini	MCP	MOON	Avg
MNIST	LeNet5	DSC	14.3	13.1	14.5	14.3	14.2	14.7	14.2	19.6	25.1	21.1	22.6	21.3	22.7	**22.1**	14.3	17.1	14.8	18.8	16.3	17.5	16.5	20.9	22.0	20.8	20.2	21.4	22.8	21.4
		NLC	243.6	281.7	256.4	237.2	224.3	286.4	254.9	243.7	370.5	238.4	159.2	219.0	392.1	270.5	210.6	334.1	201.6	135.5	184.5	342.3	234.8	72221.5	67245.1	80049.5	73114.0	71294.3	80099.9	**74004.1**
	ConvNet	DSC	13.2	14.1	13.8	13.5	13.6	14.5	13.8	19.7	22.7	19.2	20.4	21.0	23.6	21.1	20.2	20.8	20.4	21.9	20.8	23.2	21.2	21.8	23.2	22.0	19.6	21.4	23.6	**21.6**
		NLC	39.3	59.3	41.8	25.1	38.1	64.8	44.7	37.6	50.0	38.0	15.8	37.0	61.9	40.0	36.8	49.6	37.3	15.4	36.1	62.9	39.7	32942.8	25114.2	29145.5	29230.3	32638.3	37767.3	**31139.7**
SVHN	ConvNet	DSC	13.3	14.0	12.9	12.9	14.4	20.4	14.6	17.8	20.0	18.9	18.0	18.7	23.1	19.4	16.6	15.3	17.3	16.9	16.7	23.0	17.6	40.8	41.2	40.3	41.0	40.9	43.0	**41.2**
		NLC	579.4	856.6	586.6	621.6	569.8	1042.7	709.5	498.5	647.5	455.6	446.9	465.2	694.4	534.6	97.7	141.6	101.7	100.9	100.2	177.5	119.9	52887.2	51019.8	54202.4	53893.3	64288.7	70153.6	**57740.8**
	VGG	DSC	14.1	14.3	15.2	14.8	14.5	15.5	14.7	22.8	23.7	22.7	20.4	22.1	23.4	22.5	17.5	17.7	17.9	13.3	17.1	17.6	16.8	40.5	42.2	41.0	38.1	40.3	42.0	**40.7**
		NLC	175.1	197.8	181.7	146.4	176.1	200.3	179.6	94.9	142.9	100.6	65.4	102.2	131.3	106.2	77.6	80.1	80.8	42.0	79.3	82.5	73.7	2097.6	2162.7	2046.0	2173.2	2100.7	2186.0	**2127.7**
CIFAR10	ConvNet	DSC	21.7	22.3	21.8	22.4	22.0	22.6	22.1	26.3	27.0	25.5	23.0	26.4	27.1	**25.9**	22.2	23.8	22.9	22.9	22.5	23.4	23.0	18.9	19.1	18.1	18.2	18.3	18.6	18.5
		NLC	19.1	20.4	21.3	22.0	21.5	22.2	21.1	14.1	28.6	13.5	3.4	9.5	8.3	12.9	6.3	8.1	8.0	6.1	6.2	7.4	7.0	19834.5	17582.8	19243.6	17043.8	18810.1	19485.7	**18666.7**
	VGG	DSC	14.6	15.8	15.3	15.3	14.4	15.9	15.2	23.3	24.6	23.3	21.3	23.3	24.9	**23.4**	14.5	14.0	15.7	13.5	14.8	14.9	14.6	19.4	19.6	20.4	18.9	19.3	19.6	19.6
		NLC	453.7	569.2	455.4	361.2	459.1	461.0	460.0	327.7	452.6	314.9	277.3	342.0	463.1	362.9	128.3	133.8	152.2	51.2	127.7	143.4	122.8	74601.5	78328.8	77688.2	77987.3	77217.1	78727.2	**77425.0**
CIFAR100	VGG	DSC	19.9	20.8	20.1	19.9	20.7	20.6	20.3	24.5	25.7	23.9	24.0	24.6	25.1	**24.6**	20.9	19.2	20.9	12.6	21.2	21.0	19.3	22.7	22.7	22.9	22.1	23.2	23.6	22.9
		NLC	469.6	476.8	460.8	425.8	473.6	486.0	465.5	420.9	484.4	411.7	389.7	417.5	459.7	430.7	555.8	571.1	530.9	338.3	572.7	590.8	526.6	95856.8	108587.4	98737.9	94458.0	95540.6	110974.9	**100692.6**
	ResNet	DSC	11.5	10.9	11.5	11.4	11.8	11.9	11.5	20.4	20.4	20.2	17.6	19.1	20.7	**19.7**	13.5	14.2	14.0	13.2	14.0	14.4	13.9	19.4	19.3	19.4	18.9	19.5	19.5	19.3
		NLC	34.6	35.2	34.5	32.1	35.0	35.9	**34.5**	33.1	36.1	32.9	29.5	31.9	34.6	33.0	32.0	33.8	30.8	8.3	28.2	34.0	27.9	31.9	32.3	32.2	32.4	32.8	33.4	32.5
Avg Rank		DSC	3.50	3.50	3.88	3.38	3.88	3.63	3.67	1.63	1.38	1.50	1.50	1.63	1.50	1.67	3.13	3.25	2.75	3.13	2.88	3.00	2.83	1.75	1.88	1.88	2.00	1.63	1.88	1.83
(I)		NLC	2.13	2.63	2.00	2.00	2.00	2.38	2.00	2.88	2.50	3.00	3.13	3.13	2.88	3.00	3.63	3.63	3.75	3.88	3.75	3.38	4.00	1.38	1.25	1.25	1.00	1.13	1.38	1.00
Avg Rank		DSC	21.50	19.63	19.63	20.38	20.38	15.88	-	8.25	3.75	8.38	9.25	6.75	3.00	-	17.38	15.75	14.38	16.13	15.00	11.75	-	9.50	7.38	9.13	11.88	9.13	6.50	-
(II)		NLC	12.25	9.00	11.38	14.88	11.88	7.50	-	15.25	9.50	16.13	21.13	17.25	11.13	-	19.88	15.88	18.75	23.63	20.50	14.38	-	5.88	5.63	5.50	5.25	5.13	2.25	-
# Top-1		DSC	0	0	0	0	0	0	0	4	5	5	4	3	4	5	0	0	0	1	0	1	0	4	3	3	4	5	3	3
(I)		NLC	1	0	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	7	7	7	8	7	7	7
# Top-1		DSC	0	0	0	0	0	0	-	0	1	0	0	0	4	-	0	0	0	0	0	0	-	1	1	0	0	0	1	-
(II)		NLC	0	0	0	0	0	0	-	0	1	0	0	0	0	-	0	0	0	0	0	0	-	1	0	0	0	0	6	-

[1] For each test input generation approach, the highest coverage result across six seed selection strategies is underlined. The best average results 'Avg' among the four test input generation approaches are highlighted in bold.
[2] Uncolored cells indicate instances where SUNTest achieves significantly higher coverage than the baselines under the same seed selection strategy. Gray cells denote where SUNTest performs worse than or shows no statistically significant difference from the baseline.

observations demonstrate that SUNTest significantly outperforms three state-of-the-art approaches in terms of triggering diverse fault types in most of the investigated DNN models.

As highlighted by the 'Avg Rank (II)' results, SUNTest paired with MOON outperforms other generators, while SUNTest paired with Deepgini achieves the second-best performance. Specifically, for DLFuzz and ADAPT, the DeepGini strategy achieves the lowest average ranks compared to other seed selection strategies. For RobOT and SUNTest, MOON achieves the lowest average ranks, and Deepgini performs the second best. Several factors may explain these results. DeepGini tends to generate the highest number of error-revealing test inputs, which naturally increases the likelihood of finding various fault types. MOON not only considers error-revealing test inputs but also promotes the diversity of test inputs, which helps enhance the variety of erroneous behaviors.

5.2.3 Coverage Analysis. Compared to DLFuzz and RobOT, SUNTest achieves significantly higher average DSC across nearly all the investigated DNN models, except for CIFAR10-ConvNet. Compared to ADAPT, SUNTest obtains remarkably higher average DSC on the SVHN dataset, with values of 41.2 and 40.7. ADAPT achieves the highest average DSC results on the CIFAR10 and CIFAR100 datasets. According to the 'Avg Rank (I)' results, ADAPT achieves the best average rank of 1.67, compared to the average ranks of 1.83, 2.83, and 3.67 obtained by SUNTest, RobOT, and DLFuzz, respectively. In the experiments, we observed that SUNTest tends to generate test inputs with higher DSA values compared to DLFuzz, ADAPT, and RobOT. DSC is computed by counting how many buckets within the pre-defined DSA value ranges are covered by a test suite. Therefore, several inputs generated by SUNTest yield high DSA values beyond the pre-defined ranges (*i.e.*, $[L, U]$). These test inputs do not contribute to an increase in DSC.

NLC is more indicative of a diverse test suite, as a diverse test suite is likely to uncover different neuron correlations. As can be observed, SUNTest achieves remarkably higher NLC than state-of-the-art approaches across all seed selection strategies and investigated DNN models. For instance, for CIFAR10-ConvNet, SUNTest reaches an average coverage value of 18,666.7 across six seed selection strategies, while DLFuzz, ADAPT, and RobOT achieve average values of 21.2, 12.9, and 7.0, respectively. According to the 'Avg Rank (I)' results, SUNTest ranks first, followed by DLFuzz, ADAPT, and RobOT. These rankings are consistent with the 'Avg Rank (I)' results for the number of the triggered fault types shown in Table 5. This finding indicates that NLC is correlated with the diversity of fault types, which conforms to the finding observed by the literature [78]. The superior performance of SUNTest in coverage increase can be attributed to the following reasons. 1) Test inputs generated using various mutation operators (pixel-level and image-level operators) are more diverse than those generated by adding perturbations to specific pixels. 2) SUNTest is more effective in detecting diverse fault types compared to baseline approaches.

As highlighted by the underlined results, MOON outperforms other seed selection strategies in achieving higher DSC and NLC. BE ranks second in performance. These observations can be attributed to several factors. MOON employs a customized multi-objective optimization algorithm that optimizes both diversity and error-revealing capability to guide the search process. BE relies on the concept of FOL, which quantifies how unfamiliar the input is to the DNN model. By selecting seeds with both low and high FOL values, BE ensures that the inputs include both familiar and unfamiliar examples. A set of inputs ranging from those similar to the training set (with low DSA values) to those very different from the training set (with high DSA values), results in a high DSC. Both MOON and BE select seed inputs with a wider range of SA values, facilitating the subsequent mutated test inputs to achieve a higher DSC. Similarly, both strategies are designed to select seed inputs that induce diverse neuron correlations or activations within the network. This promotes better exploration of the model's internal representations, leading to higher NLC values.

5.2.4 Input Validity. Table 7 presents a comparison of IS and FID scores achieved by different test input generators. A higher IS score indicates that the generated inputs are both diverse and of high quality. A lower FID score suggests that the generated inputs are more similar to real images, both in terms of visual quality and distribution.

Table 7. Comparison of Test Input Generators in Terms of Input Validity

Dataset	IS				FID			
	DLFuzz	ADAPT	RobOT	SUNTest	DLFuzz	ADAPT	RobOT	SUNTest
MNIST	1.30	1.32	1.39	4.22	3.68	4.43	10.72	53.42
SVHN	1.23	1.33	1.76	3.24	2.53	3.67	34.35	67.66
CIFAR10	1.15	1.16	1.62	3.53	2.13	2.41	34.47	68.86
CIFAR100	1.12	1.10	1.33	2.98	4.23	4.60	46.44	70.15

SUNTest achieves the highest IS scores across all four datasets, which signifies that the inputs generated by SUNTest are both diverse and of high quality. However, SUNTest also obtains the highest FID scores across the datasets. Several factors may explain these findings. SUNTest employs higher-order mutation operators, whereas other methods typically rely on pixel-level operators. While these higher-order mutation operators could preserve the semantic content of the samples, they can increase the distance between the mutated inputs and the original seeds. This effect is particularly noticeable with transformations such as translation, rotation, and brightness adjustment, which modify the spatial or visual properties of the generated inputs. Consequently, the increased distance between the mutated samples and the original seeds contributes to the higher FID scores. Although the FID values achieved by SUNTest are higher than other baselines, they are still at an acceptable

level. The IS metric rewards inputs that are diverse and exhibit clear features, even if these inputs are distant from the original seed. In summary, SUNTest generates test inputs that exhibit a certain level of image quality.

> Answer to RQ2: SUNTest significantly outperforms state-of-the-art approaches in generating error-revealing test inputs and triggering diverse fault types across almost all investigated DNN models. Specifically, SUNTest increases the average number of detected faults by 100.3 to 310.5, with an average improvement rate of up to 162.9%. The increase in the average number of triggered fault types ranges from 29.2 to 80.9, achieving an average improvement rate of up to 98.8%.
> Among all 24 combinations of seed selection strategies and input generation strategies, the combination of DLFuzz and Deepgini performs the best in generating error-revealing inputs, ranking first most frequently. Regarding fault diversity, SUNTest paired with MOON achieves the best performance, ranking first five times across the eight investigated models.

5.3 Answer to RQ3: Effectiveness in Model Enhancement

Table 8 presents detailed results of the average accuracy improvements of DNN models after retraining with test inputs generated by various approaches. The 'Avg_1' columns show the average performance across six different seed selection strategies for each test input generator. The results reveal that as the proportion of adversarial samples in the evaluation set increases, the model's accuracy improvement becomes more pronounced. For example, for CIFAR100-VGG, accuracy improvements range from 1.66% to 4.48% when the data compound ratio of the evaluation set is '80%+20%'. When the evaluation set consists entirely of adversarial samples ('0%+100%'), accuracy improvements rise significantly, ranging from 17.42% to 22.03%. Regarding the performance of the four test input generators, with the exception of the two models on the MNIST dataset, SUNTest achieves the best 'Avg_1' results across the majority of DNN models, as indicated by the cells highlighted with a yellow background. This outperformance is especially evident when the evaluation set contains a higher proportion of adversarial samples. For MNIST-LeNet5 and MNIST-ConvNet, while DLFuzz yields the highest accuracy improvements, the improvements of DLFuzz and SUNTest are comparable. For instance, the 'Avg_1' result achieved by SUNTest when the data compound ratio is '0%+100%' is 33.15%, which is slightly lower than DLFuzz's result of 34.55%. In comparison to ADAPT and RobOT, SUNTest demonstrates significantly higher accuracy improvements across nearly all DNN models and data compound ratios. These results highlight that SUNTest consistently outperforms other test input generators in enhancing the accuracy of DNN models, particularly when the evaluation set contains a higher proportion of adversarial samples.

Table 9 presents the average accuracy improvements across five data compound ratios, represented as the 'Avg_2' results. Additionally, for each DNN model, the '↑' rows report the average increase in accuracy improvements achieved by SUNTest compared to the baseline approaches using the same seed selection strategies. The '↑ (%)' rows present the relative improvement percentages when comparing SUNTest to other baselines. As observed from the results, with the exception of the two models on the MNIST dataset, SUNTest obtains the best 'Avg_2' results across the four test input generators. Specifically, for MNIST-LeNet5 and MNIST-ConvNet, DLFuzz achieves the highest accuracy improvement, followed by SUNTest, RobOT, and ADAPT. For models trained on the SVHN, CIFAR10, and CIFAR100 datasets, SUNTest achieves the highest accuracy improvement of 10.68%, 10.15%, 7.35%, 12.15%, 11.88%, and 14.10%, respectively, with DLFuzz following behind. Considering the '↑' results, SUNTest increases the average accuracy improvements by up to 6.51%, 1.32%, 6.11%, 3.39%, 3.02%, 8.04%, 1.44%, and 6.92% on the investigated models compared to baseline approaches. As indicated by the '↑ (%)' results, these increases correspond to relative improvement rates of 67.1%, 6.4%, 133.8%, 50.0%, 69.6%, 195.8%, 13.8%, and 96.4%,

Table 8. Accuracy Improvement (%) of DNN Models after Retraining with the Generated Test Inputs

Model	Ratio	DLFuzz							ADAPT							RobOT							SUNTest						
		Random	BE	KM	DeepGini	MCP	MOON	Avg_1	Random	BE	KM	DeepGini	MCP	MOON	Avg_1	Random	BE	KM	DeepGini	MCP	MOON	Avg_1	Random	BE	KM	DeepGini	MCP	MOON	Avg_1
MNIST-LeNet5	80%+20%	6.81	6.55	6.55	6.87	6.73	6.78	**6.71**	2.91	3.59	2.91	3.77	3.03	3.75	3.33	3.39	4.97	3.60	5.75	3.72	5.77	4.53	6.54	6.66	6.51	6.58	6.59	6.68	6.59
	60%+40%	13.16	12.54	12.71	13.07	13.03	13.08	**12.93**	5.36	6.97	5.44	7.02	5.85	7.13	6.29	6.41	9.54	6.81	10.98	7.06	10.91	8.62	12.01	12.37	12.20	12.38	12.22	12.38	12.26
	40%+60%	19.18	18.33	18.64	18.99	19.01	19.06	**18.87**	7.88	10.14	7.74	10.20	8.56	10.27	9.13	9.45	13.57	9.65	15.64	10.17	15.35	12.31	16.51	16.76	16.44	16.77	16.59	16.77	16.64
	20%+80%	25.53	24.46	24.76	25.31	25.39	25.44	**25.15**	10.61	13.48	10.54	13.71	11.53	13.64	12.25	12.54	18.32	13.08	21.11	13.67	21.06	16.63	19.90	20.65	20.17	20.74	20.38	20.66	20.42
	0%+100%	31.42	28.97	30.69	30.99	31.45	31.55	**30.85**	15.35	20.26	15.49	18.04	15.35	20.43	17.49	18.90	20.01	18.68	20.95	18.93	20.97	19.74	24.81	25.21	25.01	25.33	25.14	25.22	25.12
MNIST-ConvNet	80%+20%	11.42	11.16	11.33	11.68	11.47	11.65	**11.45**	10.72	10.66	10.77	10.63	10.71	10.81	10.72	10.89	10.89	10.81	10.80	10.80	10.93	10.85	11.17	11.51	11.13	11.62	11.20	11.21	11.31
	60%+40%	16.79	16.15	16.58	17.20	16.92	17.08	**16.79**	15.50	15.46	15.55	15.44	15.49	15.58	15.50	15.72	15.71	15.58	15.60	15.56	15.80	15.66	16.49	17.06	16.44	17.21	16.56	16.74	16.75
	40%+60%	22.61	21.74	22.30	23.21	22.82	22.63	**22.55**	20.78	20.76	20.85	20.78	20.81	20.87	20.81	21.11	21.09	20.93	20.94	20.89	21.15	21.02	21.03	21.83	20.81	22.05	21.04	21.81	21.43
	20%+80%	28.25	27.10	27.89	29.28	28.50	28.54	**28.26**	25.96	25.91	26.04	25.98	26.04	26.07	26.00	26.32	26.23	26.08	26.06	26.08	26.13	26.15	26.82	27.69	26.42	28.02	26.77	27.62	27.22
	0%+100%	35.23	34.97	34.89	32.36	34.06	35.79	**34.55**	30.33	30.01	30.17	30.42	29.92	30.48	30.22	31.24	31.03	31.24	30.70	31.17	31.46	31.14	32.66	33.40	32.43	34.01	32.54	33.87	33.15
SVHN-ConvNet	80%+20%	2.02	1.63	1.77	1.78	2.09	2.09	**1.90**	2.06	1.43	2.12	1.85	1.79	1.83	1.85	1.71	0.92	1.60	1.75	1.81	1.24	1.50	0.88	1.39	0.95	1.07	1.20	1.25	1.12
	60%+40%	3.50	2.78	3.17	3.04	3.62	3.55	3.28	4.00	2.86	4.04	3.79	3.71	3.99	3.73	3.16	1.53	2.87	3.13	3.32	3.22	2.87	5.78	6.26	5.85	5.98	5.87	5.90	**5.94**
	40%+60%	5.57	4.18	4.88	4.74	5.34	5.52	5.04	6.11	4.43	6.24	5.51	5.61	6.32	5.70	4.67	2.20	4.12	4.61	4.95	4.59	4.19	10.49	10.86	10.77	10.77	10.77	11.44	**10.85**
	20%+80%	7.22	5.37	6.33	6.12	7.05	7.55	6.61	8.30	5.89	8.55	7.54	7.52	8.54	7.72	6.47	2.92	5.75	6.18	6.28	6.59	5.70	15.16	15.45	15.27	15.45	15.39	16.08	**15.47**
	0%+100%	12.41	10.85	12.44	12.24	12.80	12.94	12.28	13.58	11.32	14.56	13.77	13.55	14.52	13.55	9.12	5.23	8.93	9.31	9.44	9.48	8.58	19.60	19.91	19.78	19.88	19.84	21.16	**20.03**
SVHN-VGG	80%+20%	4.73	4.34	4.70	4.47	4.64	4.69	4.60	3.20	3.60	3.21	4.06	2.89	4.00	3.49	4.32	4.40	4.36	3.95	4.23	4.30	4.26	4.74	4.71	4.77	4.65	4.68	4.73	**4.71**
	60%+40%	7.14	6.49	7.21	6.76	7.07	7.14	6.97	4.68	5.16	4.62	6.09	4.49	6.05	5.18	6.45	6.59	6.53	5.68	6.20	6.34	6.30	7.41	7.37	7.48	7.32	7.41	7.48	**7.41**
	40%+60%	9.47	8.56	9.41	8.91	9.26	9.37	9.16	6.05	6.69	5.96	7.99	5.85	8.03	6.76	8.37	8.60	6.44	7.37	8.10	8.22	7.85	10.04	9.91	10.08	9.86	9.95	10.06	**9.98**
	20%+80%	11.62	10.42	11.60	11.00	11.57	11.93	11.36	7.40	8.17	7.17	9.84	7.27	9.91	8.29	10.23	10.62	10.44	8.89	9.90	10.36	10.07	12.82	12.56	12.78	12.58	12.63	13.09	**12.74**
	0%+100%	13.61	13.38	14.07	13.76	13.89	14.25	13.83	9.09	9.86	8.76	12.02	8.82	12.08	10.11	12.41	13.01	12.87	11.25	12.35	12.61	12.42	15.84	15.71	16.10	15.72	15.85	16.29	**15.92**
CIFAR10-ConvNet	80%+20%	2.77	2.71	2.81	2.11	2.74	2.71	**2.64**	1.85	1.85	1.72	1.75	1.62	1.69	1.75	2.08	2.08	2.04	2.07	2.03	2.28	2.06	1.94	2.07	1.98	1.92	2.00	2.00	1.99
	60%+40%	5.18	5.33	5.33	4.29	5.05	5.24	**5.07**	3.34	3.45	3.31	3.18	3.16	3.27	3.28	3.67	3.72	3.82	3.59	3.68	3.66	3.69	3.91	3.94	4.18	3.89	3.43	3.41	3.79
	40%+60%	7.44	7.68	7.55	5.90	7.16	7.57	7.22	4.44	4.28	4.39	4.51	4.01	4.59	4.37	5.09	5.14	5.17	4.47	4.98	5.17	5.00	7.28	7.76	7.66	7.74	7.73	7.78	**7.66**
	20%+80%	9.87	10.00	9.99	7.78	9.75	10.11	9.58	5.66	5.40	5.41	5.63	5.29	5.77	5.53	6.41	6.22	6.64	6.11	6.37	6.68	6.41	10.26	10.10	10.07	11.28	9.25	10.50	**10.24**
	0%+100%	12.73	12.47	12.39	9.86	11.56	13.18	12.03	6.83	6.51	6.67	6.92	6.42	7.16	6.75	7.74	7.53	7.74	7.20	7.67	7.83	7.62	13.06	12.91	13.26	13.20	12.61	13.52	**13.09**
CIFAR10-VGG	80%+20%	3.31	3.30	3.33	3.32	3.50	3.91	3.45	2.01	2.22	1.94	2.73	1.47	1.44	1.97	3.24	3.31	3.25	3.44	3.14	3.49	3.31	3.99	3.92	3.93	3.83	4.08	4.23	**4.0**
	60%+40%	5.72	5.85	5.85	5.96	6.11	5.96	5.91	3.03	3.33	2.85	4.25	2.74	3.91	3.35	4.47	4.51	4.45	4.65	4.26	4.64	4.50	7.72	7.83	7.83	7.89	7.99	8.0	**7.88**
	40%+60%	7.74	7.57	8.06	8.10	8.05	8.09	7.94	3.80	4.03	3.58	5.59	3.73	4.47	4.20	5.63	5.74	5.88	5.97	5.57	6.17	5.83	11.73	11.81	11.91	11.91	12.01	12.59	**11.99**
	20%+80%	9.29	9.11	9.46	9.94	9.59	9.76	9.52	4.64	4.88	4.31	6.77	4.91	5.53	5.17	6.79	6.81	6.93	7.02	6.62	7.40	6.93	16.13	16.32	16.34	16.31	16.41	16.71	**16.37**
	0%+100%	11.22	10.39	10.99	12.10	11.09	11.53	11.22	5.48	5.62	4.61	7.36	5.46	6.51	5.84	7.45	7.51	7.14	7.38	7.30	8.11	7.48	20.21	20.39	20.37	20.47	20.41	21.20	**20.51**
CIFAR100-VGG	80%+20%	4.47	4.33	4.52	4.47	4.63	4.46	**4.48**	3.98	3.89	4.02	4.02	3.31	2.80	3.67	4.51	4.47	4.49	4.31	4.42	4.08	4.38	1.68	1.40	1.65	1.22	1.65	2.38	1.66
	60%+40%	7.73	7.75	7.88	7.76	7.86	7.94	7.82	6.94	7.01	6.87	7.17	6.67	6.69	6.89	7.96	7.77	7.92	7.65	7.97	7.78	7.84	6.65	6.62	6.89	6.53	6.57	6.53	6.63
	40%+60%	11.08	11.14	11.03	10.94	11.26	11.33	11.13	10.31	10.29	10.10	10.32	9.98	10.02	10.17	11.35	11.12	11.35	11.13	11.37	11.37	**11.28**	11.71	11.64	11.87	12.06	11.89	12.71	11.98
	20%+80%	14.90	15.03	15.15	15.04	15.26	15.18	15.09	13.88	13.97	13.94	14.22	13.95	14.32	14.05	15.49	15.24	15.44	15.07	15.73	16.31	15.55	16.70	16.67	16.94	17.14	17.13	18.10	**17.11**
	0%+100%	18.49	18.50	18.54	18.32	18.63	19.12	18.60	17.15	17.27	17.35	17.60	17.26	17.89	17.42	18.57	18.67	18.79	18.55	19.21	19.93	18.95	22.15	21.20	21.58	22.02	21.94	23.25	**22.03**
CIFAR100-ResNet	80%+20%	4.71	4.51	4.55	4.26	4.43	4.71	4.53	2.63	3.06	2.94	2.89	3.00	2.35	2.81	4.65	4.81	4.57	4.81	4.68	4.35	4.64	5.18	5.09	5.05	4.98	5.22	5.39	**5.15**
	60%+40%	7.30	6.77	6.95	6.36	6.81	7.76	6.99	4.53	4.89	4.74	4.97	5.07	4.90	4.85	7.20	7.24	7.03	7.21	7.07	7.51	7.21	9.29	9.30	9.15	9.19	9.38	9.76	**9.34**
	40%+60%	9.82	9.17	9.42	8.95	9.54	10.63	9.59	6.81	6.94	6.82	7.07	7.05	7.08	6.96	9.80	9.86	9.73	9.79	9.60	10.34	9.85	13.61	13.72	13.66	13.59	13.73	14.34	**13.78**
	20%+80%	12.19	11.76	12.21	11.55	12.08	12.72	12.09	9.06	8.97	9.08	9.50	9.51	9.69	9.30	12.62	12.55	12.54	12.60	12.33	13.26	12.65	17.98	18.12	18.47	18.33	18.04	19.35	**18.38**
	0%+100%	15.09	14.70	15.40	14.55	15.25	15.78	15.13	12.00	11.69	11.87	12.23	11.86	12.17	11.97	15.33	15.71	15.56	15.57	15.16	15.86	15.53	23.56	23.65	23.56	23.84	23.69	24.82	**23.85**

[1] The best 'Avg_1' result among the four test input generation approaches is highlighted in bold and displayed with a yellow-colored background.
[2] Uncolored baseline cells indicate instances where SUNTest obtains significantly higher accuracy improvements compared to the baselines under the same seed selection strategy. Gray cells indicate instances where SUNTest performs worse than or shows no statistically significant difference from the baseline.

respectively. Besides, when compared to baseline approaches, SUNTest achieves increases in average accuracy improvements of up to 4.86%, 8.04%, and 6.54% across the investigated DNN models, respectively.

Table 10 provides the 'Avg Rank' and '# Top-1' results for different test input generators. The definitions of these metrics are consistent with those in Tables 5 and 6. In Table 10, both 'Avg Rank' and '# Top-1' metrics are calculated based on the accuracy improvements achieved across all data compound ratios. As indicated by the 'Avg Rank (I)' results, SUNTest is the most effective in guiding model enhancement. It achieves the best average rank of 1.53, followed by DLFuzz, RobOT, and ADAPT, with values of 1.98, 2.80, and 3.70, respectively. One interesting finding is that the ranking of these four approaches in terms of retraining effectiveness is consistent with their ranking in terms of error-revealing test input generation. Regarding the effectiveness in triggering errors, SUNTest and DLFuzz consistently rank among the top two approaches, while RobOT and ADAPT occupy the bottom two positions. These findings suggest that retraining with the generated error-revealing inputs leads

Table 9. Average Accuracy Improvement (%) of DNN Models Across Five Data Compound Ratios

Model		DLFuzz							ADAPT							RobOT							SUNTest						
		Random	BE	KM	DeepGini	MCP	MOON	Avg_1	Random	BE	KM	DeepGini	MCP	MOON	Avg_1	Random	BE	KM	DeepGini	MCP	MOON	Avg_1	Random	BE	KM	DeepGini	MCP	MOON	Avg_1
MNIST-LeNet5	Avg_2	19.22	18.17	18.67	19.04	19.12	19.18	**18.90**	8.42	10.89	8.42	10.55	8.86	11.04	9.70	10.14	13.28	10.36	14.89	10.71	14.81	12.37	15.95	16.33	16.07	16.36	16.18	16.34	16.21
	↑	-3.27	-1.84	-2.60	-2.68	-2.94	-2.84	-2.70	7.53	5.44	7.64	5.81	7.32	5.30	6.51	5.81	3.05	5.70	1.47	5.47	1.53	3.84	-	-	-	-	-	-	-
	↑ (%)	-17.0	-10.1	-13.9	-14.1	-15.4	-14.8	-14.3	89.4	50.0	90.7	55.1	82.6	48.0	67.1	57.3	23.0	55.0	9.9	51.1	10.3	31.1	-	-	-	-	-	-	-
MNIST-ConvNet	Avg_2	22.86	22.22	22.60	22.75	22.76	23.14	**22.72**	20.66	20.56	20.68	20.65	20.59	20.76	20.65	21.06	20.99	20.93	20.82	20.90	21.10	20.97	21.63	22.30	21.45	22.58	21.62	22.25	21.97
	↑	-1.23	0.08	-1.15	-0.16	-1.13	-0.89	-0.75	0.98	1.74	0.77	1.93	1.03	1.49	1.32	0.58	1.31	0.52	1.76	0.72	1.15	1.01	-	-	-	-	-	-	-
	↑ (%)	-5.4	0.3	-5.1	-0.7	-5.0	-3.8	-3.3	4.7	8.5	3.7	9.4	5.0	7.2	6.4	2.7	6.2	2.5	8.5	3.5	5.5	4.8	-	-	-	-	-	-	-
SVHN-ConvNet	Avg_2	6.14	4.96	5.72	5.58	6.18	6.33	5.82	6.81	5.19	7.10	6.49	6.43	7.04	6.51	5.02	2.56	4.65	4.99	5.16	5.02	4.57	10.38	10.77	10.53	10.63	10.61	11.17	**10.68**
	↑	4.24	5.81	4.81	5.05	4.43	4.84	4.86	3.57	5.59	3.42	4.14	4.18	4.13	4.17	5.36	8.21	5.87	5.64	5.45	6.14	6.11	-	-	-	-	-	-	-
	↑ (%)	68.9	117.1	84.1	90.3	71.7	76.4	83.5	52.4	107.7	48.2	63.8	64.9	58.6	64.1	106.6	320.8	126.2	112.8	105.7	122.2	133.8	-	-	-	-	-	-	-
SVHN-VGG	Avg_2	9.31	8.64	9.40	8.98	9.29	9.48	9.18	6.08	6.70	5.94	8.00	5.86	8.01	6.77	8.36	8.64	8.13	7.43	8.16	8.37	8.18	10.17	10.05	10.24	10.03	10.11	10.33	**10.15**
	↑	0.85	1.41	0.84	1.05	0.82	0.85	0.97	4.08	3.35	4.30	2.03	4.24	2.32	3.39	1.81	1.41	2.11	2.60	1.95	1.96	1.97	-	-	-	-	-	-	-
	↑ (%)	9.2	16.4	9.0	11.6	8.8	9.0	10.6	67.1	50.1	72.3	25.3	72.4	28.9	50.0	21.7	16.3	26.0	35.0	23.9	23.5	24.1	-	-	-	-	-	-	-
CIFAR10-ConvNet	Avg_2	7.60	7.64	7.61	5.99	7.25	7.76	7.31	4.42	4.30	4.30	4.40	4.10	4.50	4.34	5.00	4.94	5.08	4.69	4.95	5.13	4.96	7.29	7.35	7.43	7.61	7.01	7.44	**7.35**
	↑	-0.31	-0.29	-0.18	1.62	-0.25	-0.32	0.05	2.87	3.06	3.13	3.21	2.91	2.95	3.02	2.29	2.42	2.35	2.92	2.06	2.32	2.39	-	-	-	-	-	-	-
	↑ (%)	-4.1	-3.7	-2.4	27.1	-3.4	-4.1	0.6	64.8	71.1	72.8	73.0	70.9	65.5	69.6	45.8	48.9	46.2	62.2	41.6	45.2	48.2	-	-	-	-	-	-	-
CIFAR10-VGG	Avg_2	7.46	7.24	7.54	7.88	7.67	7.85	7.61	3.79	4.02	3.46	5.34	3.66	4.37	4.11	5.52	5.58	5.53	5.69	5.38	5.96	5.61	11.95	12.05	12.08	12.08	12.18	12.55	**12.15**
	↑	4.50	4.81	4.54	4.20	4.51	4.70	4.54	8.16	8.04	8.62	6.74	8.52	8.18	8.04	6.44	6.48	6.55	6.39	6.80	6.59	6.54	-	-	-	-	-	-	-
	↑ (%)	60.3	66.4	60.2	53.2	58.8	59.8	59.7	215.3	200.2	249.0	126.2	232.5	187.1	195.8	116.7	116.2	118.4	112.4	126.4	110.5	116.6	-	-	-	-	-	-	-
CIFAR100-VGG	Avg_2	11.33	11.35	11.43	11.31	11.53	11.60	11.42	10.45	10.49	10.45	10.67	10.24	10.34	10.44	11.58	11.46	11.60	11.34	11.74	11.90	11.60	11.78	11.51	11.79	11.79	11.84	12.59	**11.88**
	↑	0.44	0.16	0.36	0.49	0.31	0.99	0.46	1.33	1.02	1.33	1.12	1.60	2.25	1.44	0.20	0.05	0.19	0.45	0.10	0.70	0.28	-	-	-	-	-	-	-
	↑ (%)	3.9	1.4	3.2	4.3	2.7	8.5	4.0	12.7	9.7	12.7	10.5	15.6	21.8	13.8	1.7	0.5	1.6	4.0	0.8	5.9	2.4	-	-	-	-	-	-	-
CIFAR100-ResNet	Avg_2	9.82	9.38	9.71	9.13	9.62	10.32	9.67	7.01	7.11	7.09	7.33	7.30	7.24	7.18	9.92	10.03	9.88	10.00	9.77	10.27	9.98	13.92	13.97	13.98	13.99	14.01	14.73	**14.10**
	↑	4.10	4.59	4.27	4.85	4.39	4.41	4.44	6.92	6.86	6.89	6.65	6.71	7.49	6.92	4.00	3.94	4.09	3.99	4.25	4.47	4.12	-	-	-	-	-	-	-
	↑ (%)	41.8	48.9	44.0	53.1	45.6	42.7	45.9	98.8	96.5	97.2	90.7	92.0	103.5	96.4	40.3	39.3	41.4	39.9	43.5	43.5	41.3	-	-	-	-	-	-	-

[1] For each DNN model, among the 'Avg_1' values across the four test input generation approaches, the best 'Avg_2' result is highlighted in bold and presented with a yellow background.

Table 10. Average Rank and Top-1 Frequency of Test Input Generators for Model Enhancement

	DLFuzz							ADAPT							RobOT							SUNTest						
	Random	BE	KM	DeepGini	MCP	MOON	Avg_1	Random	BE	KM	DeepGini	MCP	MOON	Avg_1	Random	BE	KM	DeepGini	MCP	MOON	Avg_1	Random	BE	KM	DeepGini	MCP	MOON	Avg_1
Avg Rank (I)	1.90	2.18	1.98	2.08	1.90	1.88	1.98	3.68	3.68	3.68	3.55	3.73	3.68	3.70	2.83	2.75	2.80	2.88	2.78	2.88	2.80	1.60	1.40	1.58	1.55	1.60	1.58	1.53
Avg Rank (II)	8.60	12.05	9.40	10.45	8.60	6.35	-	20.00	20.65	19.95	18.05	20.85	17.60	-	14.43	14.35	15.35	15.08	15.05	12.53	-	8.05	6.53	7.65	6.35	7.33	5.00	-
# Top-1 (I)	13	8	13	12	15	14	13	1	0	1	1	0	0	0	2	2	1	0	1	0	1	24	30	25	27	24	26	26
# Top-1 (II)	3	1	1	4	1	2	-	0	0	1	0	0	0	-	0	0	0	0	1	0	-	0	1	2	2	0	21	-

to the correction of erroneous knowledge acquired by the model. Diverse fault types enable the model to learn more diverse information. Both factors contribute to the enhancement of model robustness.

Since eight DNN models were evaluated across five different data compound ratios, the total number of cases is $5 \times 8 = 40$. Regarding the 'Avg_1' results, SUNTest ranks first most frequently, appearing in the top position 26 times, which accounts for nearly two-thirds of the cases. DLFuzz ranks second, with a frequency of 13 occurrences. Regarding the results achieved by different seed selection strategies, regardless of the chosen seed selection strategy, SUNTest consistently achieves the lowest average rank among the four test input generators.

When considering the results of each approach combination ('Avg Rank (II) and '# Top-1(II)' results), the combination of SUNTest and MOON achieves the highest frequency of being ranked first among all 24 combinations, appearing in the top position 21 times. This combination also achieves the lowest average rank of 5.0. Two approach combinations, *i.e.*, SUNTest and DeepGini, and DLFuzz and paired with MOON achieve the second-lowest average rank of 6.35. Furthermore, as observed from the 'Avg Rank (II)' results, when comparing different seed selection strategies within the same test input generator, the MOON strategy outperforms the

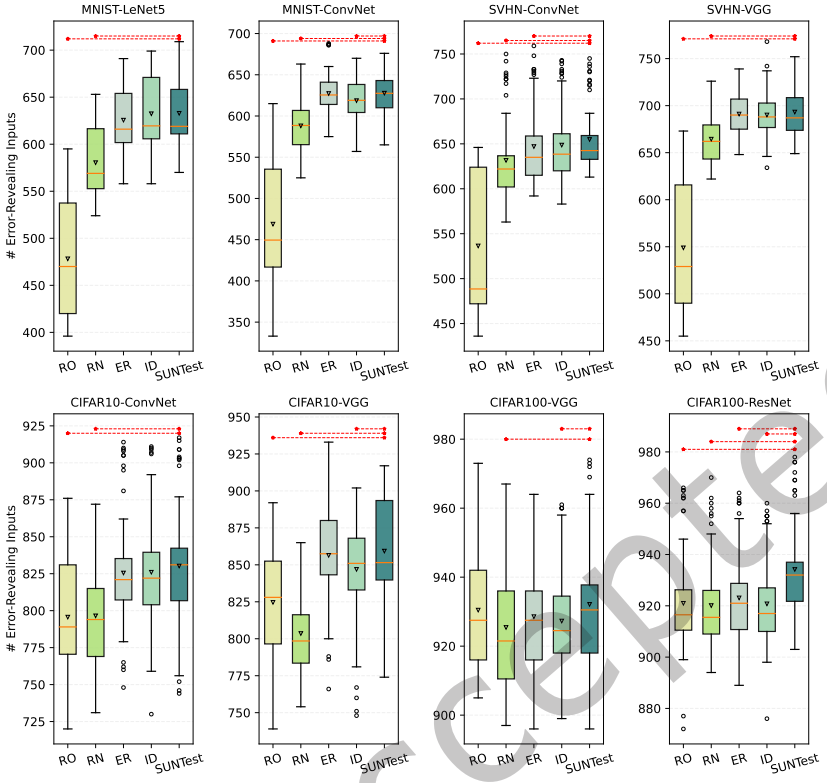

Fig. 4. Comparison Results between SUNTest and its Variants in Terms of the Number of Error-Revealing Test Inputs

others. Across all four test input generators, MOON consistently achieves the lowest average rank compared to the other selection strategies.

> Answer to RQ3: SUNTest is effective in generating test inputs for model enhancement through retraining, increasing the average accuracy improvements by up to 4.86%, 8.04%, and 6.54% compared to baseline approaches. This superior performance is attributed to its effectiveness in fault detection and its capability to trigger diverse fault types. Besides, SUNTest paired with MOON outperforms other approach combinations in enhancement, achieving the top position in nearly one-third of the cases.

5.4 Answer to RQ4: Ablation Study

Figure 4 and Figure 5 present the comparative analysis between SUNTest and its variants, specifically focusing on the number of error-revealing test inputs and the types of triggered faults across six seed selection strategies. The results are illustrated using boxplots, where the horizontal line indicates the median value, the triangle marker represents the average result, and the circle marker denotes the outlier point. For each model, red dotted lines with '★' symbols at both ends above certain pairs of boxes indicate that SUNTest significantly outperforms the corresponding variant, based on p-values and $|\delta|$ values. Table 11 presents the comparative results between

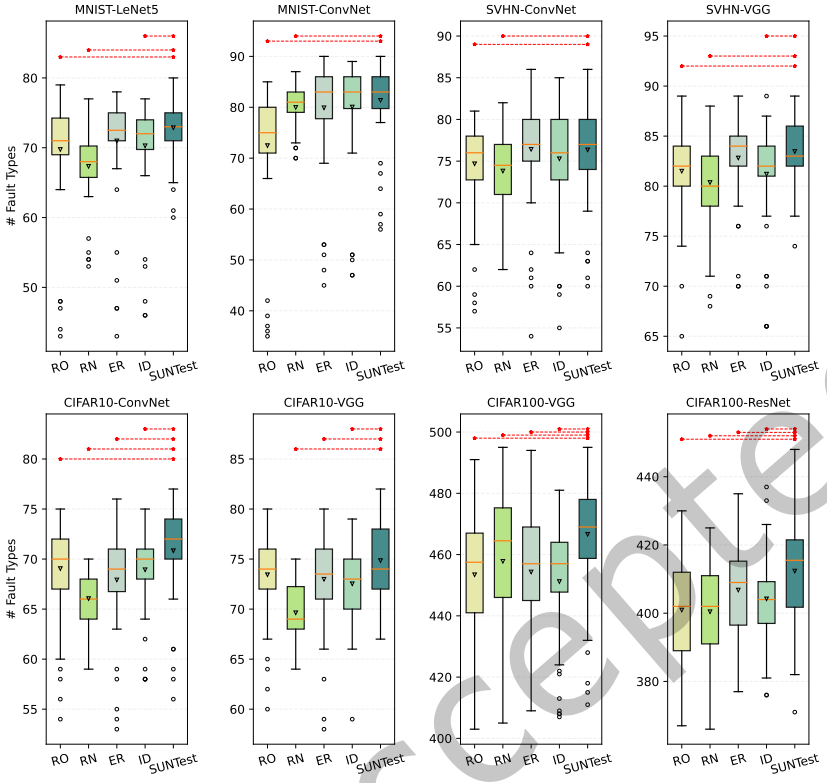

Fig. 5. Comparison Results between SUNTest and its Variants in Terms of the Number of Triggered Fault Types

SUNTest and its variants concerning model accuracy improvement. The 'Avg_1' rows report the average accuracy improvement across five data compound ratios for each approach, while the 'Avg_2' rows denote the average improvement across all eight investigated models.

5.4.1 Usefulness of the Adaptive Operator Selection Strategy. As depicted in Figure 4, the ER, ID, and SUNTest variants generate significantly larger numbers of error-revealing test inputs compared to the RO variant. This highlights the contribution of the adaptive operator selection strategy to fault detection. Regarding the capability to trigger diverse fault types, SUNTest statistically significantly outperforms the RO variant. These findings align with our expectation that considering the historical capabilities of fault detection and fault diversity triggering of mutation operators enhances the likelihood that subsequent mutated test inputs will result in a greater quantity of, and more diverse, erroneous behaviors.

In terms of retraining effectiveness, as shown in Table 11, SUNTest achieves significantly higher accuracy improvements than RO across the majority of data compound ratios (27 out of 40). This observation indicates that both fault detection and fault diversity capabilities contribute to enhancing model robustness. Therefore, we can conclude that the adaptive selection strategy for mutation operators is useful in finding more and diverse faults, and guiding model enhancement.

5.4.2 Usefulness of the Guidance from Suspicious Neurons. ER, ID, and SUNTest generate larger numbers of error-revealing inputs compared to RN. These results highlight the effectiveness of suspicious neuron guidance in

Table 11. Comparison Results between SUNTest and its Variants in Terms of the Accuracy Improvement

Dataset	Model	Ratio	RO	RN	ER	ID	SUNTest
MNIST	LeNet5	80%+20%	6.27	5.43	6.54	6.57	**6.59**
		60%+40%	11.14	10.74	12.31	**12.32**	12.26
		40%+60%	15.00	14.53	16.77	**16.78**	16.64
		20%+80%	17.88	17.78	20.61	**20.73**	20.42
		0%+100%	24.04	22.86	23.93	24.04	**25.12**
		Avg_1	14.87	14.27	16.03	16.09	**16.21**
	ConvNet	80%+20%	10.50	10.10	11.20	11.21	**11.31**
		60%+40%	16.09	14.90	16.55	16.58	**16.75**
		40%+60%	**21.45**	20.38	21.04	21.14	21.43
		20%+80%	26.19	24.37	26.65	26.74	**27.22**
		0%+100%	32.53	30.57	32.46	32.55	**33.15**
		Avg_1	21.35	20.07	21.58	21.64	**21.98**
SVHN	ConvNet	80%+20%	**2.07**	0.82	1.27	0.99	1.12
		60%+40%	5.57	5.41	**6.04**	5.84	5.94
		40%+60%	9.31	9.98	10.80	10.47	**10.85**
		20%+80%	13.07	14.58	15.44	15.22	**15.47**
		0%+100%	16.99	19.37	19.97	19.61	**20.03**
		Avg_1	9.40	10.03	**10.71**	10.43	10.68
	VGG	80%+20%	4.71	4.38	4.73	**4.73**	4.71
		60%+40%	**7.47**	7.07	7.43	7.40	7.41
		40%+60%	**10.23**	9.26	10.05	10.02	9.98
		20%+80%	12.50	11.82	**12.78**	12.76	12.74
		0%+100%	15.08	14.40	15.60	15.65	**15.92**
		Avg_1	10.00	9.39	10.12	10.11	**10.15**
CIFAR10	ConvNet	80%+20%	1.93	1.96	1.91	1.92	**1.99**
		60%+40%	**3.85**	3.49	3.55	3.55	3.76
		40%+60%	7.63	4.55	7.66	**7.67**	7.66
		20%+80%	9.55	6.44	9.48	9.42	**10.24**
		0%+100%	12.04	10.70	11.97	12.02	**13.09**
		Avg_1	7.00	5.43	6.92	6.92	7.35
	VGG	80%+20%	1.74	3.32	3.37	3.40	**4.00**
		60%+40%	6.64	7.24	7.92	**7.96**	7.88
		40%+60%	10.88	10.93	11.83	11.64	**11.99**
		20%+80%	15.66	14.77	**16.45**	16.41	16.37
		0%+100%	19.84	18.01	20.13	20.12	**20.51**
		Avg_1	10.95	10.85	11.94	11.91	**12.15**
CIFAR100	VGG	80%+20%	**1.70**	-1.31	1.51	1.35	1.66
		60%+40%	6.46	2.37	6.41	6.10	**6.63**
		40%+60%	11.17	6.62	11.12	11.10	**11.98**
		20%+80%	16.35	13.35	16.33	16.28	**17.11**
		0%+100%	20.96	19.25	21.13	21.05	**22.03**
		Avg_1	11.33	8.06	11.30	11.18	**11.88**
	ResNet	80%+20%	5.02	5.07	5.06	5.09	**5.15**
		60%+40%	9.25	9.25	9.25	9.31	**9.34**
		40%+60%	13.63	13.54	13.61	13.67	**13.78**
		20%+80%	17.85	17.59	17.79	17.89	**18.38**
		0%+100%	22.42	22.28	22.38	22.44	**23.85**
		Avg_1	13.63	13.55	13.62	13.68	**14.10**
		Avg_2	12.32	11.45	12.78	12.74	13.06
		Avg Rank	3.38	4.68	2.80	2.53	1.70
		# Top-1	6	0	3	6	25

[1] For each data compound ratio and the 'Avg_1' result, the highest values are highlighted in bold and displayed with a light purple background.
[2] Baseline cells with a gray background indicate instances where SUNTest statistically significantly outperforms its variants.

enhancing error detection. Notably, SUNTest significantly outperforms RN in generating error-revealing inputs across all investigated models. When considering the number of triggered fault types, SUNTest demonstrates significantly better performance than RN, as highlighted by the red dotted lines in the figures.

In terms of retraining effectiveness, out of a total of 40 instances, SUNTest is only found to be not significantly superior to RN in 3 cases. Additionally, as indicated by the 'Avg_2' results, the RO variant achieves the lowest average accuracy improvement, at just 11.45%, and holds an average rank of 4.68, positioning it last among the five variants. These results demonstrate that, compared to using randomly selected neurons, focusing on those identified as contributing to misbehaviors allows SUNTest to enable more targeted and effective fault detection, thereby enhancing the model's robustness.

5.4.3 Usefulness of the Hybrid Fitness Function. In terms of the number of error-revealing test inputs, SUNTest outperforms ER and ID in terms of average results across nearly all the investigated models. When compared to ER, SUNTest produces higher average results in most cases (7 out of 8), with the exception of SVHN-VGG. In terms of the triggered fault types, SUNTest outperforms the ID variant considerably in 6 out of the 8 investigated models. For models such as CIFAR10-ConvNet, CIFAR10-VGG, CIFAR100-VGG, and CIFAR100-ResNet, SUNTest triggers significantly larger numbers of fault types than both ID and ER, as indicated by the red dotted lines in figures. Furthermore, we observed that ER outperforms ID in terms of triggering more and diverse faults. Considering both fault detection and fault diversity across the eight investigated models (a total of 16 cases), ER achieves higher average and median values than ID in more than one-third of the cases (11 out of 16).

Regarding model enhancement, SUNTest achieves the highest 'Avg_2' result of 13.06%, surpassing the performance of both ER and ID, which yield results of 12.78% and 12.74%, respectively. SUNTest attains the lowest average rank of 1.70, followed by ID and ER, with values of 2.53 and 2.80, respectively. The performance of the ER and ID variants is comparable. In terms of the '#Top-1' results, SUNTest achieves the highest frequency of obtaining the greatest average accuracy improvement across all data compound ratios, followed by ID and RO, which perform as the second-best approaches. Several factors may account for the superior performance of SUNTest compared to ER and ID. First, ID evaluates whether an input should be retained as a seed for subsequent mutation solely based on its potential to increase input diversity, overlooking its ability to detect faults. Second, the ER variant focuses exclusively on the error-revealing capabilities of test inputs, which may hinder the DNN model's ability to learn from a broader range of features during the retraining process. SUNTest incorporates both fault detection and input diversity. Therefore, these findings indicate that the hybrid fitness function, which incorporates both error-revealing capability and diversity feedback of test inputs, provides better guidance for fault detection and model retraining than using individual fitness functions.

> Answer to RQ4: The adaptive mutation operator selection strategy, guidance from suspicious neurons, and the hybrid fitness function contribute to the effectiveness of test input generation. Combining them leads to improvements in fault detection and model enhancement.

5.5 Answer to RQ5: Efficiency Analysis

Table 12 reports the average execution time overhead for suspicious neuron localization and test input generation across various datasets and models.

Table 12. Average Execution Time Overhead of Suspicious Neuron Localization and Test Input Generation (Minutes)

Dataset	Model	Suspicious Neuron Localization		Test Input Generation			
		Spectrum Analysis	Suspiciousness Measurement	DLFuzz	ADAPT	RobOT	SUNTest
MNIST	LeNet5	30.70	< 0.01s	42.10	0.30	0.75	11.21
	ConvNet	48.78	< 0.01s	72.01	0.34	0.73	11.58
SVHN	ConvNet	80.35	< 0.01s	223.08	0.88	1.46	14.69
	VGG	167.16	< 0.01s	309.27	1.02	2.56	13.01
CIFAR10	ConvNet	122.66	< 0.01s	148.42	0.99	1.51	9.26
	VGG	81.65	< 0.01s	468.43	1.03	2.58	11.40
CIFAR100	VGG	98.36	< 0.01s	646.26	2.22	4.06	12.57
	ResNet	87.10	< 0.01s	969.45	2.82	5.69	14.19

5.5.1 Efficiency Analysis of Suspicious Neuron Localization. As shown in Table 12, the time complexity of suspicious neuron localization is primarily driven by the neuron spectrum analysis. The time overhead for this analysis ranges from 30.70 minutes (LeNet5 on MNIST) to 167.16 minutes (VGG on SVHN). It is largely determined by both the number of neurons in the target layer and the size of the training set. This process involves traversing the output activations of all neurons in the target layer across all training instances. Although the process is time-consuming, it is performed only once at the beginning. To mitigate computational complexity, several strategies can be employed: 1) selecting representative instances from the training set, and 2) choosing appropriate target layers to control the number of neurons. Furthermore, we plan to optimize SUNTest's suspicious neuron localization algorithm in future work to reduce computational overhead.

Once the neuron spectrum is constructed, the process of measuring neuron suspiciousness becomes computationally negligible, with execution times consistently under 0.01 seconds, regardless of the model or dataset. In practical applications, selecting an appropriate SBFL formula to compute suspiciousness is a straightforward task, even for developers working with custom datasets and models, thanks to the high efficiency of the suspiciousness calculation step. This allows practitioners to experiment with different configurations to identify the most suitable SBFL formula and the optimal neuron selection ratio for their specific use case.

5.5.2 Efficiency Analysis of Test Input Generation. It can be observed that ADAPT is the most efficient approach, demonstrating the lowest average execution time, ranging from 0.30 minutes to 2.82 minutes. RobOT follows closely behind in terms of efficiency. In the experiments, we also observed that as DLFuzz's generation process progresses, the time required to mutate each seed input increases. For instance, the mutation time for the first seed in the list may only take 1 second, while the mutation time for the 100th seed could extend to approximately 10 minutes. To alleviate this time overhead, we divided the seed list into two separate subsets for efficiency analysis. Despite this adjustment, DLFuzz remains the approach with the highest execution overhead.

Our proposed SUNTest's generation algorithm requires slightly more time compared to ADAPT and RobOT, with its average execution time ranging from 9.26 minutes to 14.19 minutes. For example, consider CIFAR100-ResNet, the most computationally intensive case in our investigated models. The overall average runtime for SUNTest is 14.19 minutes, and the time required per seed is calculated as $\frac{14.19}{200} \times 60 \approx 4.3$ seconds. The mutation process in SUNTest involves not only pixel-level mutations but also image-level operators, which makes it more time-consuming compared to ADAPT and RobOT. In general, the overhead of SUNTest's test input generation algorithm is acceptable in practice. While SUNTest is not the fastest among the approaches, it strikes a balance between computational cost and the functionality it provides. Additionally, the time overhead of SUNTest's test input generation is partly due to input mutation with various domain-specific operators, while the majority arises from the computation of the hybrid fitness function. The time cost of fitness calculation is influenced by the number of neurons in the target layer of the model. If the number of neurons can be controlled, such as by limiting the number of suspicious neurons in the target layer (only selecting those most relevant to misclassifications) or by choosing a target layer near the output layer with a manageable number of neurons, then the overall time cost can be kept within an acceptable range.

Answer to RQ5: While SUNTest is not the fastest in terms of overall time overhead for test input generation, it outperforms DLFuzz across all models. The suspicious neuron localization process is relatively more time-consuming, and future work will focus on optimizing this step to improve efficiency.

6 DISCUSSION

6.1 Implications

6.1.1 Focused Testing with Interpretation Results. This work leverages critical neuron behaviors for guiding the testing process, focusing on those involved in mispredictions. As demonstrated by the results for answering RQ1, the misbehaviors of DNN models are reduced when the signs of suspicious neurons' weights are reversed. By understanding which neurons activate in response to specific inputs, we gain deeper insights into the model's decision-making and the causes of mispredictions. Integrating neuron-based interpretation techniques into testing pipelines enables a more focused testing strategy. By identifying and targeting the neurons most critical to the model's outputs, testers can design test inputs that stress these neurons, uncovering potential weaknesses or edge cases. This ensures a thorough evaluation of the model's most impactful components.

Additionally, while this paper localizes suspicious neurons by executing training inputs, the methodology could also be directly applied to testing inputs. It is important to note that the choice of instances used in neuron spectrum analysis directly influences the calculation of suspiciousness scores. Our test input generator is motivated by the practical consideration that testing data for DNN models is often limited or incomplete. In this context, SUNTest prefers the use of training inputs to identify erroneous behaviors in the model's learned knowledge. Whether utilizing training or testing instances, the underlying idea remains the same: both strategies aim to capture internal model behaviors that reflect potential erroneous patterns. These findings can then be leveraged to further test and enhance the model's robustness.

6.1.2 Enhancing DNN Robustness with Diverse and Fault-Inducing Test Inputs. DNN models suffer from data distribution shifts, exhibiting high accuracy on the training set but failing to generalize well to inputs outside the training distribution due to natural variations in environmental conditions, such as changes in illumination and camera distortions. Results from RQ3 and RQ4 indicate that considering both the fault-revealing capability and diversity of test inputs is effective in uncovering erroneous behaviors and enhancing the robustness of DNN models. From a testing perspective, diverse test inputs reveal a greater number of faults in DNN models. It prevents the detection of redundant faults, ensuring a balanced testing process across different scenarios. Additionally, it optimizes the testing budget and effort by increasing the likelihood of discovering unique faults rather than repeatedly identifying the same issues. From a model enhancement perspective, retraining DNN models with error-inducing test inputs helps correct erroneous knowledge learned previously. Incorporating test input diversity prevents the model from disproportionately focusing on specific patterns or classes, thereby improving performance across various types of inputs. Therefore, to test and enhance DNN models, we recommend developers and testers consider both the fault-revealing power and diversity of test inputs.

6.1.3 Demonstration of the Generalizability of SUNTest. To assess the generalizability of SUNTest across different domains, we apply it to transformer-based models trained on the IMDB dataset [51], demonstrating its effectiveness in localizing suspicious neurons within text classification models. The IMDB dataset is a widely recognized benchmark for binary sentiment classification, consisting of 50,000 movie reviews labeled as either positive or negative. It is commonly used to evaluate the performance of Natural Language Processing (NLP) models, particularly in sentiment analysis tasks. In this study, three sentiment classification models are trained, each built on a pre-trained transformer-based language model (BERT) [12] as the backbone. These models employ distinct classification architectures: (1) a Recurrent Neural Network (RNN)-based classifier, (2) a Feedforward Neural Network (FNN)-based classifier, and (3) a Long Short-Term Memory (LSTM)-based classifier. The input text is first processed through a tokenizer, which converts raw text into tokenized representations. These tokenized inputs are then passed through the transformer model, where contextualized embeddings are extracted. Finally, these embeddings are fed into the respective classifier layers to generate sentiment predictions.

Following the experimental setup in RQ1, we employ two adversarial attack strategies for the IMDB dataset: Genetic Attack [5] and MHA [79]. These adversarial samples, combined with the original test set, form mixed input sets with three different data composition ratios. Given the large number of neurons in the investigated Transformer-based models, we constrained the range of λ and determined the optimal setting based on experimental results. Table 13 presents the inconsistency rates of Transformer-based models after flipping the weights of suspicious neurons. The results indicate that compared to the original model (where $\lambda = 0$), the IR values decrease when suspicious neurons undergo ablation. As a preliminary prototype, SUNTest is capable of localizing suspicious neurons in the text classification task and Transformer-based models.

Table 13. The Inconsistency Rates (%) of Transformer-Based Models after Flipping Suspicious Neurons

Model	**λ Configuration**	**Data Compound Ratio**		
		100% + 0%	**60% + 40%**	**40% + 60%**
Bert + RNN	$\lambda = 0$	13.2	46.3	62.8
	$\lambda = 0.04$	13.1	43.6	58.4
Bert + LSTM	$\lambda = 0$	12.7	45.6	62.4
	$\lambda = 0.04$	12.6	43.4	58.6
Bert + FNN	$\lambda = 0$	10.4	42.6	60.2
	$\lambda = 0.04$	10.2	39.7	55.1

It is important to note that SUNTest treats individual neurons as computational units for fault localization. While this approach is straightforward for Convolutional Neural Network (CNN)-based feedforward networks, applying SBFL to architectures such as RNNs and Graph Neural Networks (GNNs) requires additional considerations due to their distinct structural and computational properties. For instance, in LSTM networks, each layer is unrolled to process sequential inputs, enabling the model to capture temporal dependencies over time. This recurrent nature complicates the direct application of SBFL, as activations of internal states are influenced by previous time steps. In GNNs, computations occur on graph-structured data, where node interactions and message passing introduce additional dependencies. These dependencies must be accounted for when identifying suspicious nodes. These architectural differences necessitate modifications to SBFL techniques to ensure effective fault localization in models beyond FNN-based architectures.

6.1.4 Feasibility in Large-Scale Models. This section discusses the feasibility of applying SUNTest to large-scale models in terms of the computational complexity and the scope of testing.

(1) Computational Complexity of White-box Testing. This work adopts a white-box testing paradigm, which involves analyzing the internal structure of a model to guide the testing process. It is feasible for detecting faults in small to medium-scale models. One of the practical applications of this work lies in model compression, particularly for deploying large-scale models on mobile or embedded devices with limited computational resources. However, its applicability may diminish as model sizes increase, particularly with the emergence of large-scale models containing billions or even trillions of neurons/states. The complexity of model architectures and the vast scale of training datasets make exhaustive white-box analysis computationally impractical in real-world scenarios. The key advantage of white-box testing is its ability to leverage internal model information, particularly insights into the decision-making logic learned during training. This allows for a deeper understanding of model behavior. In contrast, black-box testing does not require access to a model's internal structure, making it more broadly applicable across different architectures. For example, black-box testing evaluates test adequacy based on a model's input domain. Given the computational constraints associated with large-scale models, black-box testing may offer a more efficient and practical alternative.

In this work, the search space is defined by the mutation operators. Specifically, SUNTest selects mutation operators from a predefined pool, prioritizing those most likely to enhance the fitness value of the mutated input. The fitness value is computed based on the outputs of neurons. By carefully managing the number of neurons involved, our method remains computationally feasible while preserving its effectiveness. To mitigate the complexity of white-box testing while maintaining its advantages, several strategies can be employed: 1) Targeted Layer Selection: Focusing on specific layers, particularly those closer to the output layer, where decision-critical features are concentrated. By prioritizing such layers, the computational cost of fitness evaluation is reduced while still capturing key information relevant to model predictions. 2) Controlled Selection of Suspicious Neurons: Restricting the number of neurons analyzed to those most strongly correlated with incorrect predictions. This targeted approach minimizes computational overhead while maintaining the effectiveness of the testing process.

(2) Testing the Capabilities of Large Models. Large models exhibit strong generalization capabilities across diverse tasks. It is inherently challenging to achieve comprehensive coverage of all functional areas through mutation-based testing. In this work, we apply mutation strategies specifically designed for image classification tasks, leveraging widely used mutation operators from the field of computer vision. Since mutation operators are inherently domain-dependent, their effectiveness is closely linked to the characteristics of the tasks under evaluation. Rather than applying mutations indiscriminately across all functionalities, a more refined approach involves domain-specific testing, focusing on key tasks relevant to the model's intended application (*e.g.*, sentiment analysis, code generation, or medical diagnosis). Moreover, for testing large-scale models, defining a clear test oracle is critical, as it determines whether a test case passes or fails. In the models we examined, the test oracle is well-defined: the model's predicted label must match the ground-truth label. However, establishing a test oracle for large generative models presents a significant challenge. Unlike classification tasks, where correctness can be directly verified against ground-truth labels, generative models produce open-ended outputs, making it difficult to define objective pass/fail criteria. Addressing this issue requires further research into effective oracle definitions capable of assessing the correctness of generated outputs across different domains. Therefore, given the vast functional space and complexity of large-scale models, exhaustive testing is often impractical. Instead, targeted testing of specific domain capabilities presents a more feasible alternative.

6.2 Extension of SUNTest

6.2.1 Fault Localization and Repair of DNN models. SUNTest localizes suspicious neurons by inputting misclassified training data without considering the class to which the input belongs. To more effectively identify specific errors, particularly class-related errors (*i.e.*, errors related to each fault type), incorporating class information into the fault localization process represents a promising direction for improving SUNTest [14]. It also facilitates the fault type-specific test input generation. However, our preliminary experiments indicate that class-level fault localization based on neuron-spectrum analysis is time-consuming. Specifically, we input training data with the same fault type into the DNN model to construct the dynamic neuron spectrum. The time required for this process reaches up to 2 hours for CIFAR10-ConvNet. Therefore, a useful direction for future research is to propose an efficient class-level fault localization for DNN models.

This paper considers the localization of intra-layer neurons to characterize certain erroneous decision-making patterns of DNN models. Given the growing complexity of modern neural networks, where interactions between neurons can collectively influence the model's behavior, it is crucial to examine more intricate neuron interactions. However, analyzing neuron activation patterns introduces several challenges. For instance, adapting the SBFL technique to localize suspicious neurons requires identifying appropriate combinations of neurons. These combinations serve as the fundamental units for characterizing activation patterns that are indicative of model decisions. Besides, determining which layers to select and how many layers to include presents an additional challenge. Given these challenges, we plan to further investigate this idea in future work, with a

focus on how different fault types might be linked to specific neuron combinations (*e.g.*, t-way interactions) that exhibit sensitivity to errors. This exploration could enhance our ability to diagnose and mitigate model failures. Specifically, by considering the 2-way combinations of neurons across two layers as activation patterns, we can leverage SBFL techniques to detect potentially suspicious activation patterns.

Furthermore, there are other fundamental faults in the dataset and DL programs, such as network structure and training settings, which can impact the model quality [6, 76]. In future work, we aim to extend the SBFL technique to explore these potential faults. We also plan to explore the fusion of multiple SBFL formulas to combine the strengths of different approaches. By integrating complementary information from various formulas, this fusion has the potential to improve fault localization accuracy. Our goal is to integrate these formulas into a unified framework, enabling adaptive formula selection based on the specific context, model, or dataset characteristics.

Suspicious neurons also provide informed decisions for developers on what to repair. Automatic repair of DNN models is another promising direction for future research. The ablation of suspicious neurons demonstrates effectiveness in preventing error propagation. Nevertheless, it may compromise the accuracy of models. While retraining is able to improve the overall accuracy of models, it may lead to regression errors [46, 77].

6.2.2 Quality-Oriented Test Input Generation. The performance of SUNTest in model enhancement is evaluated using mixed input sets that include both original test inputs and adversarial examples. It is evaluated from the perspective of model robustness enhancement. SUNTest is generic and could be extended for other model properties, such as fairness, safety, and privacy. For instance, in the context of fairness testing, major challenges include localizing faults related to fairness violations in DNN models and determining the appropriate operators for input mutation. Recent studies [20, 83] offer several solutions for fairness testing and enhancement through error-inducing neuron analysis. We plan to evaluate the performance of SUNTest in fairness testing.

6.3 Threats to Validity

External Threats to Validity. The external threats to validity primarily originate from the DNN models and datasets utilized in our experiments. To mitigate these threats, we employ four widely used image datasets. To simulate diverse testing contexts, we generate mixed input sets by incorporating adversarial examples with varying ratios. To mitigate the threat arising from the DNN models, we utilize two DNN models with different architectures for each dataset. Besides, a preliminary evaluation is conducted on three text classification models. However, this study lacks an evaluation of a broader range of diverse and complex DNN structures, such as RNNs and GNNs. This paper identifies suspicious neurons as errors within DNN models, while the definition of neurons in CNNs differs from those in RNNs and GNNs. Future work will adapt SUNTest for application to DNN models with diverse architectures and extend its use to other domains.

Internal Threats to Validity. The internal threats arise from the implementation of SUNTest as well as baseline approaches used for comparison. To mitigate these issues, we utilize publicly available implementations of baseline approaches provided by their authors. The code of SUNTest is carefully checked. Besides, the configuration of various hyperparameters in this study may introduce additional internal threats to validity. We address this concern from two aspects. 1) For suspicious neuron localization, the ratio of selected neurons is varied from 0.1 to 0.6 with an interval of 0.1 in the experiment. The experimental results demonstrate that setting the parameter λ to 0.1 is an optimal choice (as detailed in Section 5.1). Regarding the hyperparameters used to determine the activation threshold for each neuron, we follow the configuration set by DeepGuage. In future work, we plan to explore the impact of the hyperparameters on the effectiveness of suspicious neuron localization, with a particular focus on those used to determine neuron activation thresholds. 2) For test input generation, we adhere to the standard settings used in existing MGF algorithms, and all compared approaches share the same iteration-stopping criterion (set to 5 iterations). A last internal threat to validity would be the randomness involved in the study. We repeated the data simulation procedure detailed in Section 4.3 five times, resulting in five different mixed

input sets. For RQ1, RQ2, RQ4, and RQ5, we conducted the experiments five times. For retraining guidance, the retraining process was also repeated five times to mitigate the effects of randomness.

Construct Threats to Validity. Using the IR metric as the primary evaluation measure for RQ1 may pose a potential threat to the construct validity of the research. While IR provides a useful indicator of model performance, it focuses primarily on the difference between predicted and actual labels after neuron ablation, which may overlook other aspects of the model's behavior. To mitigate this threat, future work could incorporate additional evaluation metrics, such as the number of regression faults [77], to capture a broader range of model performance characteristics. Moreover, the comparison among different test input generators is based on evaluation metrics commonly used in DNN testing literature [9, 19], *i.e.*, the number of error-revealing test inputs found, the number of unique fault types identified, coverage metrics, and accuracy improvement after model retraining.

7 RELATED WORK

7.1 DNN Testing

7.1.1 Test Coverage Criteria. Test adequacy, which evaluates how thoroughly a DNN model has been tested, remains a fundamental issue in DNN testing. Inspired by white-box testing of traditional software, a growing body of coverage criteria specifically designed for DNNs has been proposed at various levels of granularity [21, 47, 48, 54, 58, 74, 78]. At the neuron level, each neuron operates as an individual computing unit, with its state classified as either 'activated' or 'inactivated'. An individual neuron is deemed 'activated' when its activation value is above a predefined threshold (*e.g.*, 0.5 and 0.75). For example, Neuron Coverage (NC) [58] quantifies the proportion of activated neurons within a target layer. DeepGuage [48] further extends NC by considering the output distributions of individual neurons derived from the training set. At the layer level, the combinations or sequences of neurons are utilized to characterize the internal behaviors of DNN models. For instance, DeepImportance [21] identifies important neurons within a target layer. Based on the training set, it groups the outputs of important neurons into a set of clusters. Subsequently, Importance-Driven Coverage (IDC) is designed to measure the degree to which the test suite covers the clusters. Neuron Path Coverage (NPC) [74] utilizes sequences of critical neuron sets, termed the Critical Decision Path (CDP), to characterize the decision-making process of a DNN model. Two instances of NPC are proposed: Structural-based Neuron Path Coverage and Activation-based Neuron Path Coverage, which respectively concern the path structure and the neuron output along CDPs.

The aforementioned coverage criteria quantify the extent to which the internal elements of a DNN model are covered by a test suite, by considering the syntactic structure of DNNs. Differently, Kim et al. [36, 37] propose Surprise Coverage (SC) criteria from the perspective of input diversity. The underlying assumption is that a good test suite should exhibit high diversity, ranging from very similar to very different inputs to those observed during training. They introduce Surprise Adequacy (SA) metrics to measure the relative novelty of a test input with respect to the training set by using neuron outputs. The SC criteria are designed to quantify the SA value ranges that a test suite covers. Additionally, Yuan et al. [78] introduce a criterion, namely Neural Coverage (NLC), which captures the divergence and correlation in distributions formed by neuron outputs. This paper adopts Distance-based SC and NLC to evaluate how the test input generation approaches increase coverage metrics. Additionally, recent works propose coverage criteria specific to Recurrent Neural Networks (RNN) [13, 30]. For example, TESTRNN [30] introduces Boundary Coverage, Step-wise Coverage, and Temporal Coverage to quantify the temporal relations exhibited by RNNs when processing test inputs.

7.1.2 Test Input Generation. MGF is designed to generate test inputs that expose errors while maximizing testing metrics. Previous MGF approaches, such as DeepXplore [58], DLFuzz [25] and ADAPT [43], are designed based on the guidance from coverage metrics. RobOT [68] utilizes testing metrics related to model robustness to design the optimization objective. Detailed information on these approaches can be found in Section 4.4.2. To evaluate

the effectiveness of SUNTest, we compare it against existing MGF approaches. Additionally, several studies adopt search algorithms to solve the test input generation problem. For example, DeepTest [65] synthesizes test images by combining image transformation operators with a greedy search algorithm for autonomous driving systems. DeepAtash [84] focuses on test input generation on input features. It takes as input the desired target feature value ranges and generates test inputs by optimizing both the sparseness of the generated inputs and their closeness to the target in the feature map. Similar to our work, DFLARE [66] and DRFuzz [77] address the fuzzing problem by formulating it as a Markov Chain and leveraging the MH algorithm to guide the selection of mutation operators. Our approach differs from these techniques in two ways. First, DFLARE focuses on the deviated behaviors of compressed models, while SUNTest targets standard models. DRFuzz aims to generate inputs that trigger regression faults, *i.e.*, inputs that originally passed the test but failed in the new version. SUNTest is designed to find diverse misbehaviors of a trained DNN model and further enhance its robustness. Second, both DFLARE and DRFuzz are black-box fuzzing approaches, whereas SUNTest follows a white-box testing paradigm. SUNTest leverages the internal behaviors of models and utilizes this information to guide the fuzzing process.

Various gradient-based adversarial attack approaches, such as FGSM [23], BIM-a, BIM-b [40], and JSMA [56], have been proposed to deceive DNN models by generating adversarial examples. Similar to MGF approaches, gradient-based adversarial attack strategies calculate gradients through an objective function and search for a malfunctioning input. Besides, the core concept of adversarial training is to enhance the robustness of DNN models by incorporating adversarial examples during the training phase. These approaches differ from CGF approaches as they aim to generate common error-revealing inputs without considering the internal logic of the model. In contrast, the emphasis of CGF approaches lies in systematically examining the model's logic and exploring various internal behaviors. The proposed SUNTest approach can be considered a variant of the CGF technique, specifically designed to generate error-revealing test inputs that are likely to activate suspicious neurons (*i.e.*, searching for test inputs that could increase the outputs of suspicious neurons). Since suspicious neurons are responsible for erroneous outputs, SUNTest is designed to generate inputs that explore the faulty decision logic of DNN models. The generated test inputs are further utilized for enhancing model robustness.

The aforementioned test input generation approaches mainly focus on enhancing model accuracy or robustness. Recent studies [17, 67, 80, 83] focus on generating test inputs aimed at detecting fairness violations of DNN models. For example, QUOTE [9], an extension of the RobOT framework, utilizes ZOL and FOL metrics to generate test inputs that expose fairness violations. AEQUITAS [67] employs a two-phase (local and global) generation framework to search the input space. It is guided by a probability distribution that indicates the likelihood of identifying a fairness-violating test input. Similar to our work, NeuronFair [83] identifies neurons responsible for biased decisions. It further utilizes the gradient of the loss function, which takes into account the outputs of biased neurons, to guide the input generation process.

7.1.3 Test Input Selection. DNN testing is typically conducted by selecting a dedicated test set from available labeled data. However, manually labeling test inputs is costly. To reduce the labeling cost, recent studies have been devoted to two primary test selection tasks. The first task aims to select a subset of test inputs for estimating the accuracy of the whole testing set [10, 45]. For example, CES [45] selects test inputs by minimizing the cross entropy between the chosen subset and the original testing set.

Another selection task aims to sample a subset of test inputs that are more likely to be misclassified by the DNN model. This subset is subsequently utilized for model retraining. For example, DeepGini [18] measures the Gini impurity of test inputs using predicted probabilities of all classes. The MCP [61] approach is designed to select test inputs close to the decision boundaries by calculating the ratio of the predicted probabilities between the first and second classes. KM and BE strategies [68] select test inputs based on their FOL values. Model uncertainty metrics could be employed for test input selection. Ma et al. [50] adopt model uncertainty metrics to select test

inputs that are more likely to induce misclassifications. Hu et al. [29] conduct an empirical study to investigate the effectiveness of existing test input selection approaches in the context of data distribution shifts. They propose a distribution-aware test selection metric, DAT, which aims to select uncertain test inputs and representative test inputs from both in-distribution and out-of-distribution sets. In addition, recent studies [4, 24] formulate the test input selection task into a search-based testing problem. For example, MOON [24] customizes a multi-objective optimization algorithm to guide the search process toward maximizing the outputs of suspicious neurons while promoting diversity in neuron behaviors. This study employs several typical test input selection approaches to select test inputs that serve as seeds for subsequent mutations, as detailed in Section 4.4.4.

7.2 DNN Fault Localization

Several fault localization techniques specifically designed for DNN models have been proposed, including identifying misbehaviors of model internal elements (*i.e.*, neurons within CNNs, states within RNNs) [15, 49, 64, 82], diagnosing incorrect training code and hyperparameter settings [6, 70], and detecting data faults [76].

Neural networks rely on internal neurons to make predictions. The faulty neurons result in erroneous outputs in an explicit way. DeepFault [15] utilizes SBFL techniques to identify suspicious neurons with a fixed neuron activation threshold. Differently, SUNTest employs dynamic neuron activation thresholds based on neuron output distributions. MODE [49] locates faulty features (neurons) using heatmap differentials between correctly classified inputs and misclassified inputs. It mainly focuses on detecting over-fitting and under-fitting bugs within one class. TRADER [64] conducts a trace divergence analysis to pinpoint faulty state dimensions of RNNs that are responsible for causing misclassifications. AI-Lancet [82] performs differential feature analysis by inputting pairs of correctly and wrongly classified images into the DNN model to identify error-inducing features. It further maps these error-inducing features to error-inducing neurons by calculating the gradients of the features with respect to neurons. Among the aforementioned approaches, the key to fault localization in DNN models lies in analyzing the behavioral differences among internal neurons when processing correctly classified and misclassified inputs. With the exception of AI-Lancet, current approaches utilize natural inputs from the training or validation set to perform differential analysis of neuron behaviors. AI-Lancet generates an input pair by removing the error-inducing regions (*i.e.*, trigger patterns reconstructed by backdoor detection approaches) from a misclassified input. Additionally, existing studies repair neuron-level errors in DNN models by adjusting neuron values, modifying neuron weights, and retraining the model. For example, DeepFault [15] and MODE [49] generate test inputs based on identified faulty neurons and correct the behaviors of DNN models by retraining. Similarly, guided by the identified faulty neurons, AL-Lancet [82] adjusts neuron weights and retrains DNN models. SUNTest optimizes DNN models by adjusting the weights of suspicious neurons and generates test inputs guided by these neurons for model retraining.

Similar to traditional software program testing, faults can also be present in the programs of Deep Learning systems. To pinpoint faults in DL programs, DeepLocalize [70] localizes faulty layers and hyperparameters by monitoring the DNN behaviors of numerical errors during the feed-forward and backward propagation phases. DeepFD [6] converts the fault localization task into a learning-based multi-label classification problem. It maps runtime features extracted from the training process into various types of faults, inappropriate loss functions, missing layers, and redundant layers. Besides, recent studies [69, 71] also focus on debugging DL libraries. Faults in the dataset fundamentally influence the quality of DNN models by affecting their internal elements. DFauLo [76] is proposed to locate data faults. It captures the differences between clean and faulty data through model mutation and extracts fault features based on the prediction behaviors of DNN models. It then adopts a logistic regression model to map fault features to suspiciousness scores for each data input.

Humbatova et al. [31] explore the relationship between fault types and their activation patterns through a large-scale empirical study. The results show that activation spectra can be utilized as features for fault prediction.

There are two key differences between their work and SUNTest. 1) While their study focuses on faults related to data quality, training parameters, and optimization settings, this work focuses on discrepancies between the model's actual output and the expected output. 2) SUNTest defines the neuron spectrum as the activation attributes of neurons across various inputs, while their approach defines the spectrum as the probability distribution of neuron activation values. Their research also provides valuable insights for future work. Specifically, SUNTest's fault localization method could be extended to address faults related to data quality, model parameters, and optimization settings. This could open new avenues for leveraging SUNTest in various fault localization scenarios.

8 CONCLUSION

This paper proposes SUNTest, a suspicious neuron-aware test input generation approach designed to detect diverse faults and enhance DNN models. SUNTest utilizes dynamic neuron thresholds to establish execution spectra for neurons, enabling the localization of suspicious neurons and guiding input mutations. SUNTest designs a hybrid feedback mechanism driven by suspicious neurons and an adaptive mutation operator selection strategy, facilitating the generation of test inputs that induce diverse fault types. Experimental evaluations conducted on eight DNN models demonstrate the effectiveness of SUNTest in fault localization. SUNTest outperforms state-of-the-art test input generators in detecting diverse faults and enhancing model robustness.

ACKNOWLEDGMENTS

We thank the anonymous reviewers of this manuscript for their valuable comments. This work is supported by the Joint Funds of the National Natural Science Foundation of China (No. U2241216), the National Natural Science Foundation of China (No. 62202223), the Natural Science Foundation of Jiangsu Province (No. BK20220881), the Open Fund of the State Key Laboratory for Novel Software Technology (No. KFKT2024B27), and the Fundamental Research Funds for the Central Universities (No. NT2024020).

REFERENCES

[1] [n. d.]. The online artifact of this paper. https://github.com/TestingAIGroup/SUNTest.
[2] Rui Abreu, Peter Zoeteweij, and Arjan J.C. van Gemund. 2007. On the Accuracy of Spectrum-based Fault Localization. In *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION (TAICPART-MUTATION 2007)*. 89–98.
[3] Uber Accident. 2018. After Fatal Uber Crash, a Self-Driving Start-Up Moves Forward. https://www.nytimes.com/2018/05/07/technology/uber-crash-autonomous-driveai.html. Accessed August 1, 2021.
[4] Zohreh Aghababaeyan, Manel Abdellatif, Mahboubeh Dadkhah, and Lionel C. Briand. 2024. DeepGD: A Multi-Objective Black-Box Test Selection Approach for Deep Neural Networks. *ACM Transactions on Software Engineering and Methodology* 33, 6 (2024), 158.
[5] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani B. Srivastava, and Kai-Wei Chang. 2018. Generating Natural Language Adversarial Examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2890–2896.
[6] Jialun Cao, Meiziniu Li, Xiao Chen, Ming Wen, Yongqiang Tian, Bo Wu, and Shing-Chi Cheung. 2022. DeepFD: Automated Fault Diagnosis and Localization for Deep Learning Programs. In *Proceedings of the 44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022*. ACM, 573–585.
[7] Marinela Capanu, Gregory A. Jones, and Ronald H. Randles. 2006. Testing for preference using a sum of Wilcoxon signed rank statistics. *Computational Statistics and Data Analysis* 51, 2 (2006), 793–796.
[8] Francisco M. Castro, Manuel J. Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. 2018. End-to-End Incremental Learning. In *Proceedings of the 15th European Conference on Computer Vision, ECCV 2018 (Lecture Notes in Computer Science, Vol. 11216)*. Springer, 241–257.
[9] Jialuo Chen, Jingyi Wang, Xingjun Ma, Youcheng Sun, Jun Sun, Peixin Zhang, and Peng Cheng. 2023. QuoTe: Quality-oriented Testing for Deep Learning Systems. *ACM Transactions on Software Engineering and Methodology* 32, 5, Article 125 (jul 2023), 33 pages.
[10] Junjie Chen, Zhuo Wu, Zan Wang, Hanmo You, Lingming Zhang, and Ming Yan. 2020. Practical Accuracy Estimation for Efficient Deep Neural Network Testing. *ACM Transactions on Software Engineering and Methodology* 29, 4 (2020), 30:1–30:35.
[11] Zhilu Chen and Xinming Huang. 2017. End-to-end learning for lane keeping of self-driving cars. In *Proceedings of the 28th IEEE Intelligent Vehicles Symposium, IV 2017*. IEEE, 1856–1860.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019*. Association for Computational Linguistics, 4171–4186.

[13] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu, and Jianjun Zhao. 2019. DeepStellar: model-based quantitative analysis of stateful deep learning systems. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2019*. ACM, 477–487.

[14] Matias Duran, Xiao-Yi Zhang, Paolo Arcaini, and Fuyuki Ishikawa. 2021. What to Blame? On the Granularity of Fault Localization for Deep Neural Networks. In *Proceedings of the 32nd IEEE International Symposium on Software Reliability Engineering, ISSRE 2021*. IEEE, 264–275.

[15] Hasan Ferit Eniser, Simos Gerasimou, and Alper Sen. 2019. DeepFault: Fault Localization for Deep Neural Networks. In *Proceedings of the 22nd International Conference on Fundamental Approaches to Software Engineering, FASE 2019 (Lecture Notes in Computer Science, Vol. 11424)*. Springer, 171–191.

[16] Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. 2017. Dermatologist-level classification of skin cancer with deep neural networks. *Nature* 542, 7639 (2017), 115–118.

[17] Ming Fan, Wenying Wei, Wuxia Jin, Zijiang Yang, and Ting Liu. 2022. Explanation-Guided Fairness Testing through Genetic Algorithm. In *Proceedings of the 44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022*. ACM, 871–882.

[18] Yang Feng, Qingkai Shi, Xinyu Gao, Jun Wan, Chunrong Fang, and Zhenyu Chen. 2020. DeepGini: prioritizing massive tests to enhance the robustness of deep neural networks. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2020*. ACM, 177–188.

[19] Xinyu Gao, Yang Feng, Yining Yin, Zixi Liu, Zhenyu Chen, and Baowen Xu. 2022. Adaptive Test Selection for Deep Neural Networks. In *Proceedings of the 44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022*. ACM, 73–85.

[20] Xuanqi Gao, Juan Zhai, Shiqing Ma, Chao Shen, Yufei Chen, and Qian Wang. 2022. Fairneuron: Improving Deep Neural Network Fairness with Adversary Games on Selective Neurons. In *Proceedings of the 44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022*. ACM, 921–933.

[21] Simos Gerasimou, Hasan Ferit Eniser, Alper Sen, and Alper Cakan. 2020. Importance-driven deep learning system testing. In *Proceedings of the 42nd International Conference on Software Engineering, ICSE 2020*. ACM, 702–713.

[22] Ian J. Goodfellow, Nicolas Papernot, and Patrick D. McDaniel. 2016. cleverhans v0.1: an adversarial machine learning library. *CoRR* abs/1610.00768 (2016). arXiv:1610.00768 http://arxiv.org/abs/1610.00768

[23] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015*.

[24] Hongjing Guo, Chuanqi Tao, and Zhiqiu Huang. 2023. Multi-Objective White-Box Test Input Selection for Deep Neural Network Model Enhancement. In *Proceedings of the 34th IEEE International Symposium on Software Reliability Engineering, ISSRE 2023*. IEEE, 521–532.

[25] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. 2018. DLFuzz: differential fuzzing testing of deep learning systems. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2018*. ACM, 739–743.

[26] Fabrice Harel-Canada, Lingxiao Wang, Muhammad Ali Gulzar, Quanquan Gu, and Miryung Kim. 2020. Is neuron coverage a meaningful measure for testing deep neural networks?. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2020*. ACM, 851–862.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*. IEEE Computer Society, 770–778.

[28] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS 2017*. 6626–6637.

[29] Qiang Hu, Yuejun Guo, Maxime Cordy, Xiaofei Xie, Lei Ma, Mike Papadakis, and Yves Le Traon. 2022. An Empirical Study on Data Distribution-Aware Test Selection for Deep Learning Enhancement. *ACM Transactions on Software Engineering and Methodology* 31, 4 (2022), 78:1–78:30.

[30] Wei Huang, Youcheng Sun, Xingyu Zhao, James Sharp, Wenjie Ruan, Jie Meng, and Xiaowei Huang. 2022. Coverage-Guided Testing for Recurrent Neural Networks. *IEEE Transactions on Reliability* 71, 3 (2022), 1191–1206.

[31] Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. 2024. Spectral Analysis of the Relation between Deep Learning Faults and Neural Activation Values. In *Proceedings of the IEEE Conference on Software Testing, Verification and Validation, ICST 2024*. IEEE, 245–256.

[32] Jiajun Jiang, Ran Wang, Yingfei Xiong, Xiangping Chen, and Lu Zhang. 2019. Combining Spectrum-Based Fault Localization and Statistical Debugging: An Empirical Study. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019*. IEEE, 502–514.

[33] James A. Jones and Mary Jean Harrold. 2005. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, ASE 2005*. ACM, 273–282.

[34] Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. 2019. Deep Neural Network Compression for Aircraft Collision Avoidance Systems. *Journal of Guidance, Control, and Dynamics* 42, 3 (2019), 598–608.
[35] Robert E. Kass, Bradley P. Carlin, Andrew Gelman, and Radford M. Neal. 1998. Markov Chain Monte Carlo in Practice: A Roundtable Discussion. *The American Statistician* 52, 2 (1998), 93–100.
[36] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding deep learning system testing using surprise adequacy. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019*. IEEE / ACM, 1039–1049.
[37] Jinhan Kim, Robert Feldt, and Shin Yoo. 2023. Evaluating Surprise Adequacy for Deep Learning System Testing. *ACM Transactions on Software Engineering and Methodology* 32, 2 (2023), 42:1–42:29.
[38] Nair Krizhevsky, Hinton Vinod, Christopher Geoffrey, Mike Papadakis, and Anthony Ventresque. 2014. The cifar-10 dataset. http://www.cs.toronto.edu/~kriz/cifar.html.
[39] Nair Krizhevsky, Hinton Vinod, Christopher Geoffrey, Mike Papadakis, and Anthony Ventresque. 2014. The cifar-100 dataset. http://www.cs.toronto.edu/~kriz/cifar.html.
[40] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2017. Adversarial examples in the physical world. In *Proceedings of the 5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net.
[41] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[42] Yann LeCun and Corinna Cortes. 1998. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/.
[43] Seokhyun Lee, Sooyoung Cha, Dain Lee, and Hakjoo Oh. 2020. Effective white-box testing of deep neural networks with adaptive neuron-selection strategy. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2020*. ACM, 165–176.
[44] Zenan Li, Xiaoxing Ma, Chang Xu, and Chun Cao. 2019. Structural coverage criteria for neural networks could be misleading. In *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results, ICSE-NIER 2019*. IEEE / ACM, 89–92.
[45] Zenan Li, Xiaoxing Ma, Chang Xu, Chun Cao, Jingwei Xu, and Jian Lü. 2019. Boosting operational DNN testing efficiency through conditioning. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2019*. ACM, 499–509.
[46] Zenan Li, Maorun Zhang, Jingwei Xu, Yuan Yao, Chun Cao, Taolue Chen, Xiaoxing Ma, and Jian Lu. 2023. Lightweight Approaches to DNN Regression Error Reduction: An Uncertainty Alignment Perspective. In *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering, ICSE 2023*. IEEE, 1187–1199.
[47] Lei Ma, Felix Juefei-Xu, Minhui Xue, Bo Li, Li Li, Yang Liu, and Jianjun Zhao. 2019. DeepCT: Tomographic Combinatorial Testing for Deep Learning Systems. In *Proceedings of the 26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019*. IEEE, 614–618.
[48] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. 2018. DeepGauge: multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018*. ACM, 120–131.
[49] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. 2018. MODE: automated neural network model debugging via state differential analysis and input selection. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2018*. ACM, 175–186.
[50] Wei Ma, Mike Papadakis, Anestis Tsakmalis, Maxime Cordy, and Yves Le Traon. 2021. Test Selection for Deep Learning Systems. *ACM Transactions on Software Engineering and Methodology* 30, 2 (2021), 13:1–13:22.
[51] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. The Association for Computer Linguistics, 142–150.
[52] Barton P. Miller, Mengxiao Zhang, and Elisa R. Heymann. 2022. The Relevance of Classic Fuzz Testing: Have We Solved This One? *IEEE Transactions on Software Engineering* 48, 6 (2022), 2028–2039.
[53] Yuval Netzer, Tao Wang, Adam Coates, A. Bissacco, Bo Wu, and A. Ng. 2011. Reading Digits in Natural Images with Unsupervised Feature Learning.
[54] Augustus Odena, Catherine Olsson, David G. Andersen, and Ian J. Goodfellow. 2019. TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019 (Proceedings of Machine Learning Research, Vol. 97)*. PMLR, 4901–4911.
[55] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2018. Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets. *IEEE Transactions on Software Engineering* 44, 2 (2018), 122–158.
[56] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2016. The Limitations of Deep Learning in Adversarial Settings. In *Proceedings of the IEEE European Symposium on Security and Privacy, EuroS&P 2016*. IEEE, 372–387.

[57] Spencer Pearson, José Campos, René Just, Gordon Fraser, Rui Abreu, Michael D. Ernst, Deric Pang, and Benjamin Keller. 2017. Evaluating and improving fault localization. In *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017*. IEEE / ACM, 609–620.

[58] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *Proceedings of the 26th Symposium on Operating Systems Principles, SOSP 2017*. ACM, 1–18.

[59] Jeanine Romano, Jeffrey D Kromrey, Jesse Coraggio, Jeff Skowronek, and Linda Devine. 2006. Exploring methods for evaluating group differences on the NSSE and other surveys: Are the t-test and Cohen'sd indices the most appropriate choices. In *annual meeting of the Southern Association for Institutional Research*. Citeseer, 1–51.

[60] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved Techniques for Training GANs. In *Proceedings of the 30th International Conference on Neural Information Processing Systems, NIPS 2016*. 2226–2234.

[61] Weijun Shen, Yanhui Li, Lin Chen, Yuanlei Han, Yuming Zhou, and Baowen Xu. 2020. Multiple-Boundary Clustering and Prioritization to Promote Neural Network Retraining. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020*. IEEE, 410–422.

[62] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs.CV]

[63] Department of Motor Vehicles State of California. 2020. Autonomous Vehicle Collision Reports. https://www.dmv.ca.gov/portal/vehicle-industry-services/autonomous-vehicles/autonomous-vehicle-collision-reports/.

[64] Guanhong Tao, Shiqing Ma, Yingqi Liu, Qiuling Xu, and Xiangyu Zhang. 2020. TRADER: trace divergence analysis and embedding regulation for debugging recurrent neural networks. In *Proceedings of the 42nd International Conference on Software Engineering, ICSE 2020*. ACM, 986–998.

[65] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018*. ACM, 303–314.

[66] Yongqiang Tian, Wuqi Zhang, Ming Wen, Shing-Chi Cheung, Chengnian Sun, Shiqing Ma, and Yu Jiang. 2023. Finding Deviated Behaviors of the Compressed DNN Models for Image Classifications. *ACM Transactions on Software Engineering and Methodology* 32, 5 (2023), 128:1–128:32.

[67] Sakshi Udeshi, Pryanshu Arora, and Sudipta Chattopadhyay. 2018. Automated directed fairness testing. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018*. ACM, 98–108.

[68] Jingyi Wang, Jialuo Chen, Youcheng Sun, Xingjun Ma, Dongxia Wang, Jun Sun, and Peng Cheng. 2021. RobOT: Robustness-Oriented Testing for Deep Learning Systems. In *Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021*. IEEE, 300–311.

[69] Zan Wang, Ming Yan, Junjie Chen, Shuang Liu, and Dongdi Zhang. 2020. Deep learning library testing via effective model generation. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2020*. ACM, 788–799.

[70] Mohammad Wardat, Wei Le, and Hridesh Rajan. 2021. DeepLocalize: Fault Localization for Deep Neural Networks. In *Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021*. IEEE, 251–262.

[71] Anjiang Wei, Yinlin Deng, Chenyuan Yang, and Lingming Zhang. 2022. Free Lunch for Testing: Fuzzing Deep-Learning Libraries from Open Source. In *Proceedings of the 44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022*. ACM, 995–1007.

[72] W. Eric Wong, Vidroha Debroy, Ruizhi Gao, and Yihao Li. 2014. The DStar Method for Effective Software Fault Localization. *IEEE Transactions on Reliability* 63, 1 (2014), 290–308.

[73] Xiaoyuan Xie, Tsong Yueh Chen, Fei-Ching Kuo, and Baowen Xu. 2013. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. *ACM Transactions on Software Engineering and Methodology* 22, 4 (2013), 31:1–31:40.

[74] Xiaofei Xie, Tianlin Li, Jian Wang, Lei Ma, Qing Guo, Felix Juefei-Xu, and Yang Liu. 2022. NPC: Neuron Path Coverage via Characterizing Decision Logic of Deep Neural Networks. *ACM Transactions on Software Engineering and Methodology* 31, 3 (2022), 47:1–47:27.

[75] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. DeepHunter: a coverage-guided fuzz testing framework for deep neural networks. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019*. ACM, 146–157.

[76] Yining Yin, Yang Feng, Shihao Weng, Zixi Liu, Yuan Yao, Yichi Zhang, Zhihong Zhao, and Zhenyu Chen. 2023. Dynamic Data Fault Localization for Deep Neural Networks. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023*. ACM, 1345–1357.

[77] Hanmo You, Zan Wang, Junjie Chen, Shuang Liu, and Shuochuan Li. 2023. Regression Fuzzing for Deep Learning Systems. In *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering, ICSE 2023*. IEEE, 82–94.

[78] Yuanyuan Yuan, Qi Pang, and Shuai Wang. 2023. Revisiting Neuron Coverage for DNN Testing: A Layer-Wise and Distribution-Aware Criterion. In *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering, ICSE 2023*. IEEE, 1200–1212.

[79] Huangzhao Zhang, Hao Zhou, Ning Miao, and Lei Li. 2019. Generating Fluent Adversarial Examples for Natural Languages. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019*. Association for Computational Linguistics,

5564–5569.

[80] Peixin Zhang, Jingyi Wang, Jun Sun, Guoliang Dong, Xinyu Wang, Xingen Wang, Jin Song Dong, and Ting Dai. 2020. White-box fairness testing through adversarial sampling. In *Proceedings of the 42nd International Conference on Software Engineering, ICSE 2020*. ACM, 949–960.

[81] Ziqi Zhang, Yuanchun Li, Yao Guo, Xiangqun Chen, and Yunxin Liu. 2020. Dynamic slicing for deep neural networks. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2020*. ACM, 838–850.

[82] Yue Zhao, Hong Zhu, Kai Chen, and Shengzhi Zhang. 2021. AI-Lancet: Locating Error-inducing Neurons to Optimize Neural Networks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, CCS 2021*. ACM, 141–158.

[83] Haibin Zheng, Zhiqing Chen, Tianyu Du, Xuhong Zhang, Yao Cheng, Shouling Ji, Jingyi Wang, Yue Yu, and Jinyin Chen. 2022. NeuronFair: Interpretable White-Box Fairness Testing through Biased Neuron Identification. In *Proceedings of the 44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022*. ACM, 1519–1531.

[84] Tahereh Zohdinasab, Vincenzo Riccio, and Paolo Tonella. 2023. DeepAtash: Focused Test Generation for Deep Learning Systems. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2023*. ACM, 954–966.

[85] Daming Zou, Jingjing Liang, Yingfei Xiong, Michael D. Ernst, and Lu Zhang. 2021. An Empirical Study of Fault Localization Families and Their Combinations. *IEEE Transactions on Software Engineering* 47, 2 (2021), 332–347.