Neuron Sensitivity Guided Test Case Selection

DONG HUANG*, The University of Hong Kong, China
QINGWEN BU*, Shanghai Jiao Tong University, China
YICHAO FU, The University of Hong Kong, China
YUHAO QING, The University of Hong Kong, China
XIAOFEI XIE, Singapore Management University, Singapore
JUNJIE CHEN, College of Intelligence and Computing, Tianjin University, China
HEMING CUI, The University of Hong Kong, China

Deep Neural Networks (DNNs) have been widely deployed in software to address various tasks (e.g., autonomous driving, medical diagnosis). However, they can also produce incorrect behaviors that result in financial losses and even threaten human safety. To reveal and repair incorrect behaviors in DNNs, developers often collect rich, unlabeled datasets from the natural world and label them to test DNN models. However, properly labeling a large number of datasets is a highly expensive and time-consuming task.

To address the above-mentioned problem, we propose NSS, Neuron Sensitivity Guided Test Case Selection, which can reduce the labeling time by selecting valuable test cases from unlabeled datasets. NSS leverages the information of the internal neuron induced by the test cases to select valuable test cases, which have high confidence in causing the model to behave incorrectly. We evaluated NSS with four widely used datasets and four well-designed DNN models compared to the state-of-the-art (SOTA) baseline methods. The results show that NSS performs well in assessing the probability of failure triggering in test cases and in the improvement capabilities of the model. Specifically, compared to the baseline approaches, NSS achieves a higher fault detection rate (e.g., when selecting 5% of the test cases from the unlabeled dataset in the MNIST&LeNet1 experiment, NSS can obtain an 81.8% fault detection rate, which is a 20% increase compared with SOTA baseline strategies).

ACM Reference Format:
Dong HUANG, Qingwen BU, Yichao Fu, Yuhao Qing, Xiaofei Xie, Junjie Chen, and Heming Cui. 2024. Neuron Sensitivity Guided Test Case Selection. *ACM Trans. Softw. Eng. Methodol.* 1, 1, Article 1 (January 2024), 32 pages. https://doi.org/10.1145/3672454

1 INTRODUCTION

Deep Neural Networks (DNNs) have become an increasingly important part of various tasks and are widely used to address a range of tasks, including autonomous driving [6], medical diagnosis [4], and machine translation [50]. The significant performance of DNN-driven software has substantially changed our daily lives. However, it has also brought

[*] Both authors contributed equally to this research.

Authors' addresses: Dong HUANG, dhuang@cs.hku.hk, The University of Hong Kong, Hong Kong, China; Qingwen BU, qwbu01@sjtu.edu.cn, Shanghai Jiao Tong University, Shang Hai, China; Yichao Fu, The University of Hong Kong, Hong Kong, China, yichao@connect.hku.hk; Yuhao Qing, The University of Hong Kong, Hong Kong, China, yhqing@cs.hku.hk; Xiaofei Xie, Singapore Management University, Singapore, xfxie@smu.edu.sg; Junjie Chen, College of Intelligence and Computing, Tianjin University, Tianjin, China, junjiechen@tju.edu.cn; Heming Cui, The University of Hong Kong, Hong Kong, China, heming@cs.hku.hk.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
Manuscript submitted to ACM

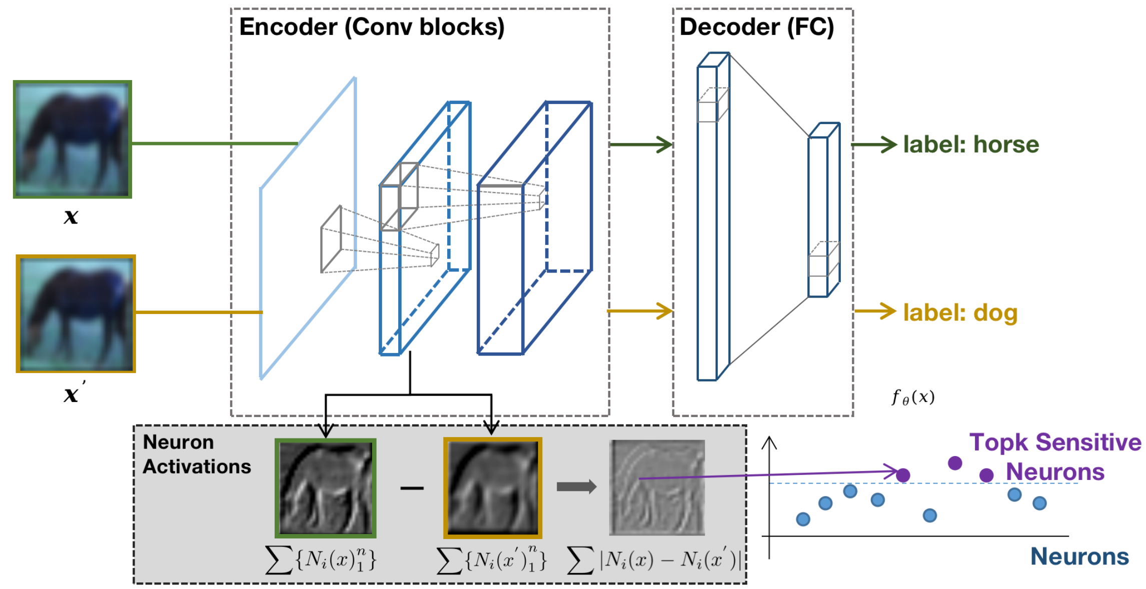


Fig. 1. *Neurons Sensitivity* is measured by the difference between the neuron activation values within the network, when the inputs are clean and mutated samples, respectively.

attention to the quality and reliability of such DNN-driven software. Like conventional software, DNN-driven software is vulnerable to defects that can result in financial losses and threaten human safety [1]. So, there is an urgent need to develop and apply effective quality assurance techniques to DNN-driven software to ensure its reliability and safety.

However, ensuring the quality of DNN-driven software is a complex issue, due to the differences between DNN models and conventional software systems [12, 13, 44]. Unlike traditional software systems that rely on developers' manual construction of logic flow, DNNs are constructed based on a data-driven programming paradigm [12, 13]. Thus, sufficient test data is critical to detect and repair incorrect behaviors of DNN-driven software [2, 5, 13, 27, 32, 70], which requires DNN developers to collect a significant amount of data from various scenarios and to hire a large workforce to label it, which is a highly expensive and time-consuming task.

Under this situation, identifying and selecting the most valuable and representative data, which will be misclassified by DNN-driven software, becomes critical for improving the effectiveness and efficiency of quality assurance tasks for DNN-driven software [12, 13, 19, 22, 32]. Inspired by the success of code coverage criteria in conventional software programs, prior researchers believe that test cases with higher neuron coverage result in higher adequacy and better quality of DNN testing. Then they proposed using neuron coverage to measure the adequacy of DNN testing [14, 36, 44]. For example, DeepXplore proposed NAC (Neuron Activation Coverage) that partitions neuron activation values into two ranges based on the threshold k, each representing a state in the DNN. NAC-guided test case selection[†] will calculate the coverage of each test case in DNN and then select tests that obtain the highest coverage in DNN, i.e., cover more state of neurons in DNN. Based on these coverage criteria, researchers have proposed coverage-guided test case selection methods [12, 14, 22, 36, 44, 63] to identify valuable tests (i.e., fault-inducing inputs that DNNs will misclassify) from candidate datasets. However, the effectiveness of these methods has been called into question by

[†] DeepGini provides the NAC-guided test case selection in Keras.

several studies [17, 31, 65], which have revealed that increased neuron coverage does not correlate with a higher Fault Detection Rate (FDR, defined as the ratio of misclassified cases among selected cases). This suggests that tests selected by neuron-coverage-guided methods may not be as valuable as initially thought. Furthermore, similar to conventional code coverage, neuron coverage also incurs a high overhead in the collection process, making it challenging to apply to large-scale models [48] and datasets [10].

To address these problems, recent researchers have proposed uncertainty-based prioritization techniques that prioritize test cases based on some rules applied to the final layer outputs of DNNs [12, 13, 32]. These techniques prioritize test cases with a high probability of detecting incorrect behaviors in DNNs, effectively collecting valuable test cases from a large unlabeled dataset. However, these prioritization techniques typically rely on the final layer outputs, which may not accurately reflect the internal neuron behaviors of the DNN and limit the visibility for the DNN developer [3, 42, 43] since focusing on the final layer cannot perceive the global information of the DNN. In many cases, DNN developers need to understand the internal logic of the DNN model and identify the root cause of incorrect behavior [33, 55, 61] to effectively optimize and debug DNN-driven software [52, 63]. Therefore, we believe that there is a need for a prioritization technique that utilizes DNN internal neuron information to help DNN developers efficiently debug incorrect behaviors.

To utilize internal neuron information with prioritization techniques, one way is to extend existing prioritization techniques (e.g., DeepGini [12], CES [32], and ATS [13]), which use the output of the final layer to calculate its metrics, to use the output of the internal neurons for prioritization directly. For example, DeepGini uses the Gini index of the output distribution produced by the DNN to prioritize test cases. An intuitive extension could involve calculating the Gini index, not just for the final output distribution, but for the outputs of internal neurons as well, and then integrating these values into the prioritization process. This approach could provide a more comprehensive view of the DNN's behavior and potentially reveal faults that are not evident from the final output distribution alone [21, 61]. However, this was found to be impractical (see Sec. 3.1) because the output of internal neurons does not carry the same semantic information as the output of the final layer, which corresponds to the confidence of each class. Therefore, a thoughtful reworking of the prioritization algorithm would require a meaningful incorporation of internal neuron information.

To address this challenge, we propose NSS (i.e., **N**euron **S**ensitivity guided test case **S**election), which is inspired by the concept of *neuron sensitivity* [21, 61, 66] in the deep neural network. As shown in Fig. 1, neuron sensitivity is a measure of how the output of a neuron changes with a small variation in its input. It is often related to the neuron's activation and is critical in understanding the neuron's response to different inputs [45, 66]. Specifically, we observe that a test case with high sensitivity values for a neuron is more likely to cause the model to produce incorrect behavior. To prioritize test cases with neuron sensitivity, we propose a novel Test case's Neuron Sensitivity Score (**TNSScore**), which represents the sum of a test case's neuron sensitivity values across all neurons in the DNN. A test case with a higher TNSScore means that it has a higher confidence in detecting incorrect behavior in DNN. However, since DNNs can have millions of neurons, calculating the TNSScore for all neurons can be computationally expensive. To address this issue and improve efficiency, we propose **Sensitive Neuron Identifier** that detects sensitive neurons in the DNN using a subset of the unlabeled dataset. Different from directly calculating TNSScore for all neurons in DNN, our approach leverages a novel algorithm (detailed in Sec. 3.3) that selects a representative subset of the dataset to accurately identify sensitive neurons, significantly reducing the computation time required. By identifying the sensitive neurons, we can reduce the computation time required to calculate the TNSScore, as we only need to calculate the TNSScore for the sensitive neurons. Thus, our approach enables the effective and efficient detection of valuable test cases with high confidence in detecting incorrect behavior in DNNs.

To validate the effectiveness of NSS, we conduct experiments with four well-designed DNN models across four widely-used datasets. We also use seven widely used data mutation strategies to generate unlabeled candidate datasets in our experiments. The experiment results demonstrate that NSS performs well in the test case selection tasks. Specifically, compared to baseline approaches, NSS obtains a higher fault detection rate (FDR). For example, when selecting a 5% test case from the unlabeled dataset in the MNIST&LeNet1 and Fashion&Resnet20 combination, NSS can obtain 81.8% and 89.4% FDR, which increases FDR from 61.8% to 81.8% for MNIST&LeNet1 combination. Using the selected test cases to retrain models can increase accuracy more than baselines. For example, when we finetune the MNIST&LeNet1 and Fashion&Resnet20 combination with 5% test cases, they can increase 9.01% and 6.79% accuracy, which is higher than baseline approaches.

In summary, we make the following contributions:

- We define the Test case Neuron Sensitivity Score (TNSScore), which can measure the confidence of a test case being misclassified by the DNN model. Then, we propose neuron sensitivity guided test case selection (NSS), which can select valuable test cases from unlabeled datasets.
- We conducted extensive experiments to investigate the performance of NSS. The results show that NSS can significantly outperform other test selection methods and efficiently enhance the DNN model.

2 BACKGROUND

2.1 Neural Network

In our work, we focus on Deep Neural Networks (DNNs) for classification, which can be presented as a complicated function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that maps an input $x \in \mathcal{X}$ to a label $y \in \mathcal{Y}$. Unlike traditional software, programmed with deterministic algorithms by developers, DNNs are defined by the training data, along with the network structures. Generally speaking, a DNN model consists of an input layer, an output layer, and at least one hidden layer. Each neuron in each layer is intertwined with neurons in other layers, and the output of each neuron is the weighted sum of the outputs of all neurons in the previous layer. Then a nonlinear activation function (e.g.,tanh, sigmoid, and ReLU) is applied. To accomplish a specified task (e.g., image classification), a model with a predefined structure often needs to be trained on a labeled dataset by solving the following optimization problem:

$$\min \mathbb{E}(\mathcal{L}(f(x), y))$$

where $\mathcal{L}$ is a loss function measuring the difference between the model output $f(x)$ and the ground-truth label y. Cross-entropy loss is normally applied for classification-oriented tasks, which is used to minimize the expected loss of the model on the training set and is usually achieved by stochastic gradient descent. However, due to the imbalance of training data, noisy labels, overfitting, and under-trained processes, DNNs also have some defects (e.g., an incorrect prediction for fault-inducing inputs). Therefore, how to fully test a DNN model becomes an important topic, especially when it is applied in some security-critical fields (e.g., autonomous driving).

2.2 Neuron Sensitivity

2.2.1 Definition. As shown in Fig. 1, neuron sensitivity in the AI community is used to quantify the change of neuron values when the input contains perturbation [61, 66]. Formally, the neuron sensitivity $S(N_i, (x, x'))$ of a neuron N_i in a DNN model on a set of dual pairs (x, x') is defined as the average L_1 norm of the differences between the outputs of neurons in the pairs of inputs, normalized by the dimension of the neuron output vector, i.e.,

$$S(N_i, (x, x')) = \left| N_i\left(x\right) - N_i\left(x'\right) \right|$$

where $N_i(x)$ is the output of neuron N_i on input x, $|\cdot|$ denotes the norm L_1 and the x' is mutated by x with DNN developers specified mutation strategies (e.g., rotation, shear, blur).

2.2.2 Neuron Sensitivity guided Fault Detection for DNNs. In the literature, several neuron-sensitivity-guided fault detection for DNNs have been proposed to detect adversarial samples [53, 66], backdoor triggers [21, 61], and Out-of-Distribution (OOD) samples [60] in DNNs. For example, ANP [61] analyzes internal neuron sensitivity based on the perturbation to analyze whether the neuron is backdoor-related. FMP [21] further utilizes feature map level sensitivity to detect backdoor feature maps in DNNs, which obtains SOTA performance in backdoor defense.

2.3 Test Case Selection from Candidate Dataset

We introduce three types of test case selection strategies that are used to select valuable tests in the candidate dataset, i.e., neuron-coverage-guided [14, 36, 44, 63], surprise-adequacy-guided [22–24], and uncertainty-guided test case selection [12, 13, 59].

2.3.1 Neuron-coverage Guided Test Case Selection. Inspired by traditional code coverage criteria, several neuron-coverage-guided test case selection methods have been proposed to select valuable tests from candidate datasets to accelerate the testing process [14, 36, 44, 63]. For example, Neuron Activation Coverage (NAC) [44] and K-multisection Neuron Coverage (KMNC)-guided test case selection [36] are pivotal in this landscape, where NAC considers a neuron covered if its activation is higher than a certain threshold. KMNC divides the activation range of each neuron into k segments, and a segment is deemed covered if the test case activations fall within it. These neuron-coverage-guided selection strategies are based on the premise that increasing the diversity of neuron activation patterns can expose more potential faults within the DNN. By prioritizing test cases that cover more unique neuron activation states or segments, developers can more effectively explore the behavior space of the model, potentially leading to the identification of faults or unexpected behaviors not captured through traditional testing methods.

2.3.2 Surprise Adequacy Guided Test Case Selection. Surprise Adequacy Guided Test Case Selection leverages the novelty or unexpectedness of inputs, identifying test cases that diverge significantly in their activation patterns from what the model has previously learned. This divergence is quantified through metrics such as Likelihood-Based Surprise Adequacy (LSA) [22] and Distance-Based Surprise Adequacy (DSA) [22], aiming to prioritize inputs that could reveal unforeseen behaviors or vulnerabilities in the DNN. Techniques like Multimodal LSA (MLSA) [24] and Multimodal MDSA (MMDSA) [23] offer refined approaches by considering the multimodal distribution of training data, thereby improving the model's exposure to a wider range of diverse and potentially challenging inputs. These methods are crucial for uncovering hidden flaws in the model by focusing on test cases that are most likely to cause the DNN to behave in unexpected ways, thereby enhancing the robustness and reliability of DNN applications.

2.3.3 Uncertainty Guided Test Case Selection. Focused on quantifying the model's confidence in its predictions, uncertainty-guided strategies [12, 13, 59] employ metrics like Vanilla Softmax, Softmax Entropy, Prediction-Confidence Score (PCS), and Monte-Carlo Dropout to rank test inputs. These metrics assess the uncertainty or confidence level associated with each prediction, targeting inputs where the DNN's output suggests a higher probability of misclassification. For example, the state-of-the-art approach, DeepGini [12], prioritizes test inputs by measuring the DNN's confidence in

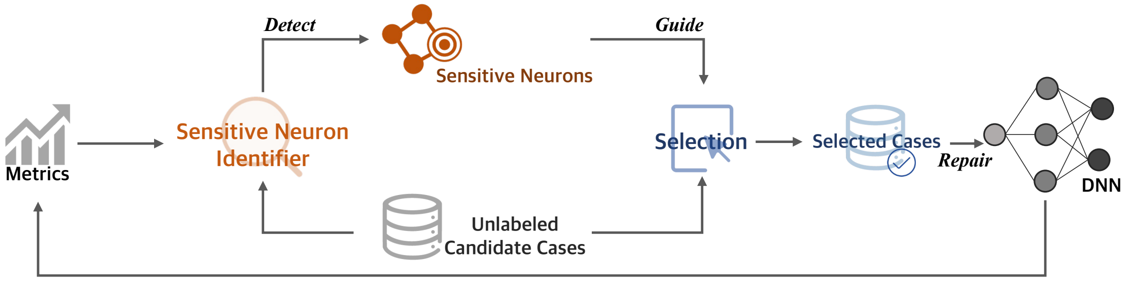


Fig. 2. Overview of NSS's workflow.

classifying each test input. More specifically, the test inputs that are predicted with more similar probabilities for all classes are prioritized higher. By prioritizing inputs with higher uncertainty, this approach aims to uncover inputs that are most informative for model improvement, particularly useful in active learning [13] to select data points that could significantly enhance the model's accuracy upon retraining.

3 METHODOLOGY

The overview of NSS's workflow is shown in Fig. 2, which mainly includes two stages: 1. detect sensitive neurons based on neuron sensitivity and 2. test case selection from the unlabeled dataset (i.e., prioritize test case by TNSScore, which is defined in Sec. 3.2). Specifically, during the testing process (Sec. 3.3), NSS first feeds the unlabeled candidate dataset into *sensitive neuron identifier* to detect sensitive neurons in the DNN model, which can reduce computational expense, and then reports sensitive neurons to DNN developers. Then the unlabeled candidate dataset will be fed into the model to calculate the Metric Score on the sensitive neurons, i.e., the Test case's Neuron Sensitivity Score (**TNSScore**). Next, NSS prioritizes test cases based on their TNSScore (Sec. 3.4). Finally, the cases with high scores will be selected by NSS.

For ease of discussion, this section defines the following notations for DNNs and neurons: f is a DNN, and there are n neurons (i.e., $\{N_i\}_1^n$) in the model. The i-th neuron in the DNN is denoted as N_i. $N_i(x)$ denotes the output of the corresponding neuron when the network input is x. The mutation case x' is generated by benign mutation strategies [†] such as rotation, blur, and scale, which are widely used for data mutation during the deep learning testing process [13, 37, 54] (See Sec. 4.1.2).

3.1 Problem when Using Prioritization in Internal Neuron Output

Recently, uncertainty-guided prioritization strategies have been widely used in test case selection in DNN models. However, these strategies rely on the final layer outputs, which may not accurately reflect the internal neuron behaviors of the DNN and limit the visibility of the DNN developers [12, 13, 32] since focusing on the final layer cannot perceive the global information of the DNN. This limitation may prevent developers from detecting incorrect behaviors in the internal neuron behaviors in DNNs [21, 40, 61, 66, 68]. To address these problems, there are two intuitive internal neuron-related prioritization strategies to utilize internal neuron information for test case selection. The first method is to extend existing prioritization techniques, such as the Gini metric [12], to use the internal neuron output for prioritization directly (internal Gini). In this section, we conduct a case study to illustrate the challenge of this approach.

[†] benign mutation in image augmentation typically refers to a minor or harmless change introduced during the augmentation process, where benign mutation could include minor alterations such as rotation, translation, scaling and cropping.

Table 1. Comparison of Fault Detection Rate (FDR) using internal Gini prioritization versus random selection for MNIST dataset with LeNet1 and LeNet5 models.

Dataset-Model	Strategy	Select 5% Test Case	Select 10% Test Case	Select 15% Test Case	Select 20% Test Case
MNIST-LeNet1	Internal Gini	18.6%	18.4%	19.5%	20.4%
	Random	21.3%	21.3%	21.3%	21.3%
MNIST-LeNet5	Internal Gini	15.0%	14.8%	16.3%	16.7%
	Random	18.8%	18.8%	18.7%	18.7%

Specifically, we evaluate the Fault Detection Rate (FDR) of the internal Gini under the MNIST dataset and LeNet1 and LeNet5 models. We calculate the Gini values of the **last encoder layer** of the LeNet1 and LeNet5 models and, based on the Gini value for each test, select the top 5%, 10%, 15%, and 20% tests from the candidate dataset. The evaluation results are shown in Tab. 1. We can observe that selecting 5% of test cases from the MNIST&LeNet1 and MNIST&LeNet5 combinations yields only 18.6% and 15.0% FDR, respectively, which is even lower than random selection (randomly selecting 5%, 10%, 15%, and 20% tests from the candidate dataset). As the test case selection ratio increases, the FDR similarly remains low or even below random selection, indicating challenges when directly applying the Gini metric to internal neuron outputs. The second method involves previously established neuron-coverage-guided test case selection (e.g., NAC-, KMNC-, and NPC-guided test case selection) [14, 36, 44, 63]. However, as indicated by prior studies [17, 31, 63, 65], neuron coverage does not correlate with Fault Detection Rate (FDR), meaning that higher neuron coverage does not guarantee higher FDR. Given the constraints of current test case selection methods, including those that utilize the Gini metric on neuron outputs and neuron-coverage-guided strategies, there is a motivation to develop more sophisticated prioritization techniques, which would leverage detailed internal neuron information to significantly enhance the effectiveness of test case selection in DNNs.

3.2 Test case's Neuron Sensitivity Score

Neuron sensitivity provides valuable information on the behavior of neurons within a DNN model and helps assess the reliability of a neuron [66]. To better prioritize test cases based on their impact on the DNN, we propose the Test case's Neuron Sensitivity Score (TNSScore). The TNSScore measures the model's sensitivity to a given test case and helps evaluate the likelihood of the test case being misclassified. Formally, given a test case x and a set of internal neurons $\{N_i\}_1^n$ in a DNN, the TNSScore of the test case x is defined as:

$$TNSScore_x = \sum_{i=1}^{n} |N_i(x) - N_i(x')| = \sum_{i=1}^{n} S(N_i, (x, x'))$$

To calculate the TNSScore, NSS first feeds the candidate test cases into the DNN to obtain the activation values for each neuron. Next, a benign mutation strategy (e.g., rotation, blur, scale) is applied to each test case to generate a mutated version x'. Finally, NSS calculates the difference between the activation values of each neuron for the original and mutated test case and uses this information to obtain the TNSScore for each test case. This metric allows developers to prioritize test cases based on their TNSScore values. Our experiments have shown that test cases with higher TNSScore are more likely to detect incorrect behavior in the DNN model.

3.3 Sensitive Neuron Identifier

Intuitively, prioritizing test cases based on the TNSScore value in all neurons is convincing. However, we notice that in the DNN model, the number of internal neurons is enormous. For example, the VGG16 model has 35,749,834 neurons, and the ResNet20 model, widely used in deep learning tasks, has 543,754 neurons, so calculating the TNSScore value in all neurons is unrealistic because it is a highly expensive and time-consuming task.

To address this problem, we propose *Sensitive Neuron Identifier* (Identifier), which is designed to identify a subset[†] of the most sensitive neurons in a given deep neural network. Using these neurons to calculate TNSScore can reduce the calculation time in the process. The detailed implementation of the *Sensitive Neuron Identifier* is provided in Alg. 1. Specifically, given a dataset of input samples, we randomly produce a set of samples from the dataset (line 3). Next, we compute the sensitivity of neurons to the selected sample pairs and record the most sensitive neurons to each pair (lines 4-10). Finally, we select the neurons found to be sensitive in most sample pairs as the final result (line 11).

Algorithm 1: Sensitive Neuron Identifier

```
Input: $f$: DNN model function
$\mathcal{X}$: Set of original test cases
$k$: Percentage of neurons to identify as sensitive
$BenignMutation$: Set of benign mutation strategies
Output: $\mathcal{SN}$: Indices of identified sensitive neurons
1  Function Identifier($f, \mathcal{X}, k$):
2      Initialize $NSList = [0] * n$ #Neuron Sensitivity List for $n$ neurons
3      $\mathcal{X}_{sub}$ = RandomSample($\mathcal{X}$, 10%) #Select 10% subset of $\mathcal{X}$
4      for $x \in \mathcal{X}_{sub}$ do
5          Randomly select a mutation $Mutation$ from $BenignMutation$;
6          $x' = Mutation(x)$;
7          Calculate activation values $\{N_i(x)\}_{i=1}^{n}$ for $f(x)$
8          Calculate activation values $\{N_i(x')\}_{i=1}^{n}$ for $f(x')$
9          for $i \in [1, \ldots, n]$ do
10             Update sensitivity: $NSList[i] += |N_i(x') - N_i(x)|$
11     Identify top $k\%$ sensitive neurons: $\mathcal{SN}$ = argsort$(NSList)[-k\% \cdot n :]$
12     return $\mathcal{SN}$
```

3.4 Prioritizing Test Case by Neuron Sensitivity

The testing and repair processes for DNN-driven systems are heavily dependent on manually labeled data. While it is often straightforward to accumulate a vast quantity of unlabeled data, the manual labeling process is considerably more costly. This issue is particularly pronounced for data that require specialized expertise for labeling, such as medical datasets, making it impractical to label all collected data without discrimination. Therefore, the selective identification of test cases becomes crucial, allowing for the prioritization of valuable data and thereby reducing labeling costs.

To select test cases with a high fault detection rate, we propose a neuron sensitivity-guided test case selection. Alg. 2 shows the workflow of our selection. It first utilizes *Sensitive Neuron Identifier* to detect sensitive neurons in DNN (line

[†] Sensitive Neuron Identifier detects sensitive neurons with a 10% subset dataset then the Cohen's Kappa Coefficients of sensitive neuron list are all above 0.95 for five different evaluation results.

Algorithm 2: NSS Test Case Selection

```
   Input: $x \in \mathcal{X}$: original test cases; $k$: Percentage of sensitive neuron detected by Sensitive Neuron Identifier; $N$:
          The number of test cases to be selected by NSS.
   output: $\mathcal{X}_f$: selected test cases
1  Function NSS $(f, \mathcal{X}, k, N)$:
2      All_TNSScore = []
3      $\mathcal{SN}$ = Identifier$(f, \mathcal{X}, k)$
4      for $x \in \mathcal{X}$ do
5          TNSScore = 0
6          Randomly select a mutation $Mutation$ from $BenignMutation$;
7          $x' = Mutation(x)$;
8          $\{N_i(x)\}_{i=1}^{n} \leftarrow f(x)$'s activation values
9          $\{N_i(x')\}_{i=1}^{n} \leftarrow f(x')$'s activation values
10         for $i \in \mathcal{SN}$ do
11             TNSScore+= $|N_i(x') - N_i(x)|$
12         All_TNSScore.append(TNSScore)
13     $\mathcal{X}_f = \mathcal{X}[argsort(\text{All\_TNSScore})[-N:]]$
14     return $\mathcal{X}_f$
```

Table 2. An example to show how NSS prioritize test cases.

Tests x, x'	$N_1(x), N_1(x'), S_1$	$N_2(x), N_2(x'), S_2$	$N_3(x), N_3(x'), S_3$	S
x_1, x'_1	0.4, 0.3, 0.1	0.5, 0.4, 0.1	0.6, 0.4, 0.2	0.4
x_2, x'_2	0.2, 0.2, 0	0.4, 0.4, 0	0.2, 0.2, 0	0
x_3, x'_3	0.4, 0.3, 0.1	0.5, 0.3, 0.2	0.3, 0.3, 0	0.3
x_4, x'_4	0.8, 0.7, 0.1	0.5, 0.45, 0.05	0.7, 0.6, 0.1	0.25
x_5, x'_5	0.5, 0.4, 0.1	0.3, 0.2, 0.1	0.4, 0.35, 0.05	0.2
x_6, x'_6	0.3, 0.25, 0.05	0.6, 0.55, 0.05	0.4, 0.4, 0	0.1

3). Then it computes the TNSScore in the sensitive neuron between each original test case x and its corresponding mutation case x' (lines 4-12). Then the test case will be prioritized by their score, i.e., the test cases with a higher score will be selected by NSS (lines 13).

Example. Here, we use a simplified example to illustrate how NSS selects test cases from unlabeled datasets. Specifically, assume that we have six different test cases x_1, x_2, x_3, x_4, x_5, and x_6 and the DNN has three sensitive neurons N_1, N_2, and N_3. We first use benign mutations (e.g., rotation, blur, scale) on the test cases to generate their corresponding mutation cases x'_1, x'_2, x'_3, x'_4, x'_5, and x'_6. Then we feed these cases into the model to obtain the neurons' activation output and neuron sensitivity. According to the value of S in Tab. 2, we can prioritize the tests as x_1, x_3, x_4, x_5, x_6, and x_2. Therefore, the DNN is most sensitive to x_1's changes, which may be due to the DNN's lack of sufficient background knowledge of x_1 and needs x_1-related knowledge to improve the DNN. While for x_2, the DNN's neuron will not be affected by x_2's mutation change, which indicates that the DNN model has sufficient knowledge about x_2, so that x_2 is not a valuable case for the DNN now (i.e., using x_2 to repair the DNN will not change the DNN's parameter).

Table 3. Datasets and DNNs for evaluating NSS, which covers the complete set of datasets evaluated by baselines.

Dataset	DNN Model	Neurons	Layers	Ori Accuracy (%)
MNIST [11]	LeNet-1 (L-1) [28]	3,350	5	89.50
	LeNet-5 (L-5) [28]	44,426	7	91.79
CIFAR-10 [25]	ResNet-20 (R-20) [18]	543,754	20	86.07
	VGG-16 (V-16) [48]	35,749,834	21	82.52
Fashion [62]	LeNet-1 (L-1) [28]	3,350	5	78.99
	ResNet-20 (R-20) [18]	543,754	20	86.12
SVHN [41]	LeNet-5 (L-5) [28]	44,426	7	84.17
	VGG-16 (V-16) [48]	35,749,834	21	92.02

Table 4. Transformations and parameters used in our experiments for generating unlabeled test cases. These mutation strategies are used for unlabeled candidate dataset generation and BenignMutation.

Transformations	Parameters	Parameter ranges
Shift	(s_x, s_y)	[0.05, 0.15]
Rotation	$q\ (degree)$	[5, 25]
Scale	$r\ (ratio)$	[0.8,1.2]
Shear	$s\ (angle)$	[15, 30]
Contrast	$\alpha\ (gain)$	[0.5,1.5]
Brightness	$\beta\ (bias)$	[0.5,1.5]
Blur	$ks\ (kernelsize)$	[2,7]

4 EVALUATION

We evaluate NSS and answer the following questions.

- RQ1 (Sensitivity): What is the correlation between TNSScore and model performance?
- RQ2 (Selection): How effective and efficient is NSS?
- RQ3 (Sample Size): How does the sample size of the sensitive neurons affect NSS's effectiveness?
- RQ4 (Layer Selection): How does the selected layer affect NSS's effectiveness?
- RQ5 (BenignMutation): How does the BenignMutation affect NSS's effectiveness?

4.1 Experiment Setup

Our evaluation was done on a GPU server with two twenty-core CPUs and four NVIDIA RTX 2080Ti graphics cards.

4.1.1 ***Datasets and Models.*** We adopt four widely used image classification benchmark datasets for the evaluation (i.e., MNIST [11], Fashion MNIST [62], SVHN [41], and CIFAR10 [25]), which are most commonly used datasets in deep learning testing [12–14, 17, 20, 22, 29, 36, 44, 54, 56, 64]. Tab. 3 presents the details of the datasets and models. The MNIST [11] dataset is a large collection of handwritten digits. It contains a training set of 60,000 examples and a test set of 10,000 examples. The CIFAR-10 [25] dataset consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. Fashion [62] is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. SVHN [41] is a real-world image dataset that can be seen as similar to MNIST (e.g., the images are of small cropped digits). The models we evaluated include LeNet [28], VGG [48], and ResNet [18], which are also commonly

Table 5. The parameter configuration of test case selection.

Criteria	Parameters	Parameter Config
Random	-	-
NAC	t (threshold)	0.5
KMNC	k (k-bins)	1000
NPC	α, β, k	0.7, 0.6, 1
DSA	n, up	1000, 2.0
LSA	n, up	1000, 2000
PRIMA	m, x	100, 10
Softmax	None	None
Dropout	Sample	200
Gini	None	None
ATS	None	None
NSS	k (percentage of sensitive neuron)	10(%)

used in deep learning testing tasks [12–14, 17, 20, 22, 29, 36, 44, 54, 56, 64]. For each dataset, we follow the setup of DeepGini [12] and ATS [13] by selecting two different DNN models. While well-trained DNNs achieve high accuracy on original test sets [49], this leads to a lack of fault-inducing inputs necessary for evaluation, leading to significantly lower FDR, making it challenging to evaluate the performance of the testing approach effectively. To address this issue, DeepGini utilizes adversarial tests for evaluation, where DNNs have low accuracy for adversarial tests. However, as mentioned in several studies [26, 31, 39, 43, 47], adversarial tests may not accurately represent data encountered in real-world scenarios. ATS tackles this problem by employing large parameters with benign mutations. Nevertheless, we observe that excessively increasing the parameter strengths can cause the mutated image to lose essential information present in the original image, potentially compromising the evaluation's validity. To address this issue, we follow the practice of Arachne [49] and BET [57] and use under-trained DNNs for evaluation. By employing under-trained DNNs, we ensure that the dataset contains sufficient misclassified samples, enabling a more comprehensive and meaningful assessment of the testing methods.

4.1.2 ***Benign Test Case Generation.***

We follow the prior benign data simulation [13, 37, 54] strategies to generate realistic unlabeled datasets. Specifically, we use seven widely used benign mutations (i.e., shift, rotation, scale, shear, contrast, brightness, and blur) to generate the test case with its original label. The parameters of the mutation are shown in Tab. 4. We do not choose adversarial attack (e.g., FGSM, PGD, and BIM) to generate test cases because these data cannot represent the data collected from the real-world scenario and can lead to unreliable conclusions [26, 31, 39, 43, 47]. During test case generation, we randomly select one benign data augmentation from our seven augmentations to mutate a test case with its original label for each test data in the dataset. When the original test size is 10,000, we will generate test cases of the same size. We show one example for each dataset to illustrate the mutated examples with each mutation strategy in Fig. 3.

4.1.3 ***Test Case Selection.***

Multiple test case selections have been proposed by recent works (e.g., neuron-coverage-guided [14, 36, 44, 54, 63], uncertainty-guided [12, 13, 46, 58], robust-guided [7, 38, 56], surprise-adequacy-guided [22]). Among these, robustness-guided selection methods have been extensively discussed [7, 26, 31, 38, 39, 43, 47, 56, 67], with a particular focus on balancing model accuracy and robustness. These studies indicate that while strategies like RobOT [56] and PACE [7] enhance robustness, they might compromise accuracy. Consequently, we will exclude these

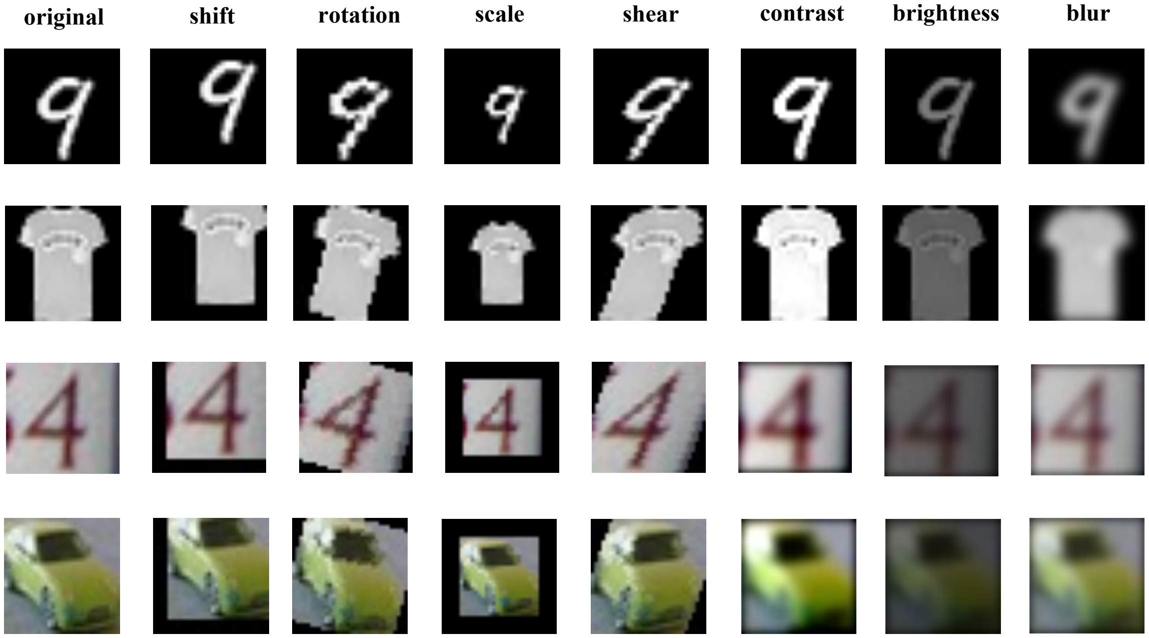


Fig. 3. Examples of unlabeled candidate dataset generated by our seven mutation strategies. The first column is original images. Then from left to right partitions are generated by shift, rotation, scale, shear, contrast, brightness, and blur mutations.

robustness-oriented methods from our baseline strategies. To compare NSS with neuron-coverage-guided test case selection, we compare NSS with the most famous metric, i.e., Neuron Activation Coverage (NAC) [44] and K-multisection neuron coverage (KMNC) [36] that was proposed by DeepXplore [44] and DeepGauge [36]. Besides, we also compare NSS with NPC, the SOTA neuron-coverage-guided test case selection proposed by Xie et al. [63], which is used to evaluate the covered path in DNN decision flow. To compare NSS with uncertainty-guided test case selection, we compare NSS with PRIMA [58], ATS [13], DeepGini [12], Vanilla Softmax [59], and MC Dropout [59], where Vanilla Softmax and MC Droput were proposed by Weiss and Tonella [59], and these strategies obtain SOTA performances compared with other uncertainty-guided test case selection methods (e.g., PCS and Entropy), so that we do not evaluate NSS with these sub-optimal methods. To compare NSS with surprise-adequacy-guided test case selection methods, we compare NSS with DSA [22] and LSA [22], these methods also illustrate SOTA performances compared with other surprise-adequacy methods, e.g., MDSA, MLSA, and MMDSA [24]. Finally, we also use Random Selection (RS) as a natural baseline, which can help us assess whether a selection method is effective.

The parameters of the selection strategies are shown in Tab. 5. Specifically, we set the threshold of NAC as 0.5 following the previous studies [12, 44, 63]. For KMNC, we follow the configuration of Ma et al. [36] and set the k value as 1000. For NPC, we follow the configuration of reference [63] and set α to 0.7. For DSA and LSA, we follow the configuration of Kim et al. [22] and set the upper bound (ub) as 2,000 and 2.0, respectively. We also set the number of buckets n as 1000. For PRIMA [58], we set the number of model mutants m to be 100 and the percentage of neurons/weights selected x to be 10 for model mutation. For MC Dropout, we follow the configuration of Weiss and Tonella [59] and use a very large number of samples (200) based on the recommendation. Other uncertainty strategies such as DeepGini do not require hyperparameters.

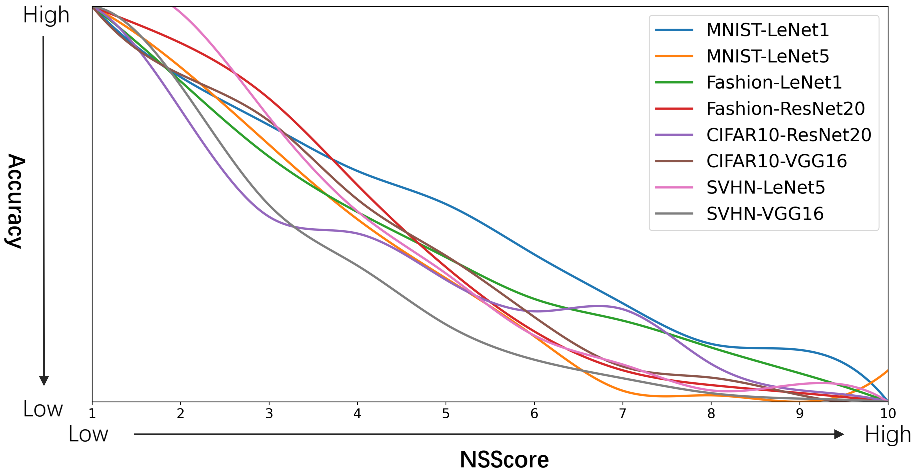


Fig. 4. Accuracy distribution under different TNSScore partitions, where from the left to right the TNSScore will increase.

Table 6. Correlation analysis of TNSScore and DNN accuracy. We provide Pearson and Spearman analysis for Fig. 4.

Dataset	MNIST		Fashion		CIFAR10		SVHN	
Model	L-1	L-5	L-1	R-20	V-16	R-20	L-5	V-16
Pearson	-0.993	-0.939	-0.975	-0.963	-0.950	-0.972	-0.952	-0.921
Spearman	-1.000	-0.927	-1.000	-1.000	-0.988	-1.000	-0.988	-1.000

4.2 What is the correlation between TNSScore and model accuracy?

To evaluate the correlation between TNSScore and model accuracy, we use the eight dataset-model combinations in Tab. 3 to calculate the TNSScore. For the BenignMutation function, we use seven different mutation strategies (i.e., shift, rotation, scale, shear, contrast, brightness, and blur) and then with different mutation strategies randomly selected from the parameter ranges in Tab. 4 to calculate each unlabeled data sample's TNSScore. The distribution of model accuracy under different TNSScore with eight combinations of models and datasets are illustrated in Fig. 4, where we first partition the dataset into ten partitions based on their TNSScore and then calculate the accuracy for each partition. We can observe that for each different model and dataset combination, all of them have the same trend, i.e., lower TNSScore will have higher accuracy, which means that TNSScore can be a representation of the test cases' confidence for correct prediction.

Correlation analysis. To illustrate the correlation of TNSScore and model accuracy for the input tests, we provide the Pearson and Spearman correlation analysis in Tab. 6 based on the TNSScore and model accuracy for the tests in different partitions in Fig. 4. Notably, all the model and dataset combinations exhibit correlation scores lower than -0.921, indicating a strong negative correlation between TNSScore and model accuracy. This suggests that as the TNSScore increases, the model accuracy tends to decrease, and vice versa. For example, the SVHN dataset combined with the VGG-16 model architecture shows a Pearson correlation score of -0.921, highlighting a significant inverse linear relationship. Similarly, the MNIST and LeNet-5 combination achieves a Spearman correlation score of -0.927.

Answer to RQ1: TNSScore is strongly negatively correlated with model accuracy for input tests. For example, the Pearson and Spearman correlation score of MNIST&LeNet1 obtains -0.993 and -1.000 for TNSScore and accuracy.

4.3 How effective and efficient is NSS?

4.3.1 Fault Detection. Similar to traditional software testing [15, 30, 69], test case selection tries to find valuable test cases from a large pool of candidate unlabeled test set, which can reduce the cost of manual labeling time once the labeling resource is limited. For a given selection method, if a selected test set can trigger more faults means that it could reveal more defects in the software. In this paper, we follow the definition of ATS [13] to utilize **F**ault **D**etection **R**ate (FDR) as our fault detection metric to measure the effectiveness of NSS's test case selection method. Specifically, the FDR is defined as follows:

$$FDR(X) = \frac{|X_{wrong}|}{|X|}$$

where $|X|$ denotes the size of the selected test cases, and $|X_{wrong}|$ is the number of test cases misclassified by DNN.

The evaluation results are shown in Tab. 7, where we compared the FDR of NSS and our baselines at different test case selection rates (5%, 10%, 15%, and 20%). We can observe that first neuron-coverage-guided test case selection methods (i.e., NAC, KMNC, NPC) have lower performance in fault detecting. Sometimes their FDRs are very similar to random selection results, which indicates that neuron coverage is not a proper metric for guided test case selection, which is consistent with previous research [12, 13, 17, 31, 58, 59]. Second, we also compare NSS with surprise-adequacy-guided test case selection methods in Tab. 7, where we can observe that compared with DSA and LSA, NSS still obtains SOTA performance in all experiments. For example, when we select 5% test cases from the candidate dataset in MNIST&LeNet1 combination, NSS obtains 81.8% FDR while DSA and LSA only obtain 22.2% and 43.6% FDR, respectively. We can also observe that surprise-adequacy-guided test case selection methods are even lower than uncertainty-guided methods in all experiments in Tab. 7, which is consistent with previous study Gao et al. [13], Weiss and Tonella [59].

Finally, we can also observe that although uncertainty-guided test case selection methods obtain SOTA performance compared with neuron-coverage-guided methods, they are still less effective compared with NSS. For example, we can observe that although DeepGini has a higher FDR compared with neuron-coverage-guided selection methods (e.g., DeepGini obtains 61.8% FDR in MNIST & LeNet1 combination when we select 5% tests in candidate dataset, while neuron-coverage-guided selection only obtains 42.2% FDR, respectively), it is still lower than NSS (e.g., when selecting 20% of the test samples, NSS increases FDR by 20 percent points compared with DeepGini in the most advantageous combination Fashion & ResNet20, even the lowest improvement is an FDR gain of 6.15% in MNIST & LeNet1). We can also observe that other uncertainty-guided test case selection methods such as PRIMA [58], Vanilla Softamax Weiss and

Table 7. Fault Detection Rate of NSS and baselines. We use green color to highlight the maximum fault detection rate. Softmax means Vanilla Softmax and Dropout means MC Droput.

Dataset(DNN)	Select 5% Test Cases											
	NSS	NAC	KMNC	NPC	DSA	LSA	PRIMA	Softmax	Dropout	Gini	ATS	RS
MNIST (L-1)	81.8	22.6	42.2	20.2	22.2	43.6	64.7	60.5	62.0	61.8	58.7	21.3
MNIST (L-5)	81.8	18.4	25.8	21.6	22.6	37.1	54.2	55.7	57.8	58.2	60.2	18.8
Fashion (L-1)	70.2	32.7	35.9	31.0	31.1	40.5	50.3	58.2	55.3	57.8	53.3	31.2
Fashion (R-20)	89.4	21.1	23.7	30.5	24.1	43.2	54.8	70.3	68.2	55.0	40.3	26.3
SVHN (L-5)	52.7	29.1	28.8	31.0	31.1	41.7	60.1	53.5	54.7	53.2	47.8	28.8
SVHN (V-16)	77.3	21.5	16.2	18.9	24.5	36.4	51.9	59.7	56.3	53.0	55.3	16.0
CIFAR-10 (V-16)	68.6	28.4	16.4	30.6	43.6	40.3	60.5	62.5	59.7	60.2	62.1	30.9
CIFAR-10 (R-20)	71.4	28.2	24.6	26.0	27.4	38.9	48.3	60.8	55.2	50.4	50.2	28.8
Dataset(DNN)	Select 10% Test Cases											
	NSS	NAC	KMNC	NPC	DSA	LSA	PRIMA	Softmax	Dropout	Gini	ATS	RS
MNIST (L-1)	70.6	20.8	38.7	20.2	22.5	40.1	55.6	53.6	55.9	55.7	54.3	21.3
MNIST (L-5)	67.7	18.1	20.8	20.0	21.3	34.7	50.1	49.9	51.3	50.5	52.3	18.8
Fashion (L-1)	65.0	30.9	35.9	31.0	31.0	36.3	43.5	52.4	51.3	48.2	48.7	31.0
Fashion (R-20)	80.7	20.5	22.0	28.4	27.1	38.8	45.3	63.7	60.5	48.7	35.2	26.2
SVHN (L-5)	52.5	29.1	27.2	31.0	31.1	36.0	53.7	48.5	48.7	47.3	42.1	29.1
SVHN (V-16)	63.8	19.6	16.2	17.1	23.4	33.5	44.7	54.6	51.3	43.1	48.7	16.0
CIFAR-10 (V-16)	64.8	29.4	17.4	30.3	43.3	37.2	53.6	55.3	56.2	56.2	56.3	30.8
CIFAR-10 (R-20)	62.8	25.8	21.8	28.7	27.4	35.8	42.1	53.1	48.2	45.0	47.1	28.9
Dataset(DNN)	Select 15% Test Cases											
	NSS	NAC	KMNC	NPC	DSA	LSA	PRIMA	Softmax	Dropout	Gini	ATS	RS
MNIST (L-1)	58.3	21.6	35.3	20.6	23.6	37.8	47.2	50.2	51.3	50.1	47.6	21.3
MNIST (L-5)	56.7	18.5	19.3	19.3	21.0	31.5	43.2	44.2	46.7	46.2	40.2	18.7
Fashion (L-1)	59.6	30.5	34.2	31.0	31.0	33.2	40.5	48.3	45.4	44.4	45.1	31.2
Fashion (R-20)	70.5	21.1	22.2	27.3	27.8	34.9	41.7	58.2	50.9	47.2	30.3	26.2
SVHN (L-5)	51.4	29.1	27.5	30.9	31.0	32.5	48.3	45.3	42.9	41.3	38.9	28.9
SVHN (V-16)	54.3	18.6	15.5	17.0	23.5	31.1	41.8	58.2	44.6	34.7	43.2	15.9
CIFAR-10 (V-16)	63.4	27.0	17.7	30.6	42.8	34.4	48.6	48.7	52.6	51.9	50.5	30.8
CIFAR-10 (R-20)	56.7	25.6	21.5	28.8	27.2	33.9	49.7	46.9	42.7	40.6	42.7	28.9
Dataset(DNN)	Select 20% Test Cases											
	NSS	NAC	KMNC	NPC	DSA	LSA	PRIMA	Softmax	Dropout	Gini	ATS	RS
MNIST (L-1)	53.3	22.0	35.7	20.9	23.5	35.6	40.3	45.2	45.8	45.4	41.5	21.3
MNIST (L-5)	50.8	18.4	19.4	19.1	20.8	29.9	37.7	40.2	41.3	42.3	37.1	18.7
Fashion (L-1	57.3	30.3	34.6	31.1	31.0	30.7	37.9	42.9	40.2	39.4	40.1	31.3
Fashion (R-20)	62.0	21.4	22.8	26.7	27.6	31.3	36.5	50.7	43.6	42.7	28.5	26.2
SVHN (L-5)	51.1	29.1	28.3	30.9	31.0	29.8	43.7	40.3	39.5	35.4	33.1	28.9
SVHN (V-16)	43.3	18.2	15.6	16.6	23.1	30.2	38.6	52.1	40.2	28.9	40.1	15.9
CIFAR-10 (V-16)	60.7	29.8	18.3	30.7	42.3	31.8	43.9	43.9	49.2	48.5	46.1	30.8
CIFAR-10 (R-20)	54.1	26.4	20.7	28.5	26.4	30.6	44.1	41.8	38.6	35.4	36.9	28.9

Tonella [59], MC Dropout [59], and ATS Gao et al. [13] have same trend in our experiments, i.e., they are more effective compared with neuron-coverage-guided test case selection methods while less effective compared with NSS.

The primary reason behind the lower FDR of uncertainty-guided selection methods lies in their prioritization criteria, i.e., higher uncertainty with a greater likelihood of incorrect predictions. This assumption leads to the uncertainty-guided selection methods trying to select tests with higher uncertainty since they believe that these cases are more prone to prediction errors. However, this strategy inadvertently neglects test cases that, despite being classified with high confidence by the DNN, are still incorrect. These overlooked cases often involve inputs that have been manipulated to exploit the model's vulnerabilities, such as adversarial examples [16, 38] or inputs containing backdoor triggers [21, 55, 61], which can activate specific, sensitive internal neurons without necessarily reflecting uncertainty in the final layer's outputs. In contrast, NSS overcomes this limitation by conducting a more nuanced analysis of test case behavior at the internal neurons. By examining how test cases influence the activation of these neurons, NSS is capable of identifying potentially misleading inputs that might be confidently misclassified by the DNN.

Answer to RQ2.1: Test cases selected by NSS have higher FDR compared with baseline strategies in most of the experiments. For example, when we select 5% tests from the MNIST&LeNet1 combination, NSS obtains an 81.8% FDR while baselines only obtain a 64.7% FDR.

4.3.2 Fault Detection Diversity. As mentioned by ATS [13], faults in deep learning testing share similarities with traditional software testing [8, 9]. In both domains, fault-inducing inputs tend to be very dense and located close to one another, which means that two faults may reflect the same defect in DNN. To analyze the model more comprehensively, we hope the test selection methods cannot only detect more faults but also detect more diverse faults efficiently. We leverage the concept of fault type introduced in [13], which is defined as:

$$Fault\_Type(x) = (Label(x)^* \rightarrow Label(x))$$

where $Label(x)^*$ denotes the ground-truth label, and $Label(x)$ denotes the DNN prediction. For a typical classification dataset with 10 different categories, the number of possible fault types is $10 \times 9 = 90$. As the candidate test cases to be selected may not introduce all types of errors, we use the *ratio of area under the curve (RAUC)* following previous study [13, 58] to measure the diversity of each selection methods.

The evaluation results of fault diversity with the percentage of selected cases increasing from 1% to 20% are shown in Fig. 5, where we can observe that NSS achieved SOTA fault diversity compared to random selection and baseline methods under most of the dataset&model combinations when we select more than about 13% of the candidate samples, the curve of NSS is always higher than other methods. We also calculated the ratio of area under the curve (RAUC) when we selected 20% tests from the candidate dataset to show more accurately the ability of different methods to find diverse errors, as listed in Tab. 8. We can observe that first, the RAUC of neuron-coverage-guided test case selection is close and even sometimes lower than Random Selection (RS)'s results, which is consistent with previous study [13, 63]. For example, in the MNIST&LeNet1 combination, only KMNC obtained SOTA RAUC (i.e., 81.48%) compared with RS (i.e., 74.01%), while for the MNIST&LeNet5 combination, NAC, KMNC, and NPC have lower RAUC compared with RS. Then, we can also observe that the RAUC of NSS is still higher than DSA and LSA. For example, in the MNIST&LeNet1 combination, NSS obtains 88.99% RAUC, while DSA and LSA only obtain 59.77% and 62.68% RAUC, respectively. One reason is that the DSA and LSA have lower FDR compared with NSS when we select 20% tests. For example, when we select 20% tests in MNIST&LeNet combination, NSS obtains 53.3% FDR while DSA and LSA only obtain 23.5% and 35.6% FDR. Then, we can also observe that NSS also has a higher RAUC compared with uncertainty-guided baselines (i.e., DeepGini, ATS, Vanilla Softmax, MC Dropout, and PRIMA). For example, in MNIST&LeNet1 combination, the RAUC

Table 8. When selecting 20% test cases, the RAUC of fault type coverage rate plots. L means LeNet, R means ResNet, and V means VGG.

Dataset	MNIST		Fashion		CIFAR10		SVHN	
Model	L-1	L-5	L-1	R-20	V-16	R-20	L-5	V-16
NAC	57.36	55.49	54.12	46.71	58.81	65.53	69.75	47.15
KMNC	81.48	55.86	81.57	52.63	65.39	64.71	88.35	47.98
NPC	50.68	61.29	52.78	63.15	62.07	65.07	65.61	49.84
DSA	59.77	56.46	60.72	70.45	77.03	68.27	63.83	67.02
LSA	62.68	57.56	64.41	71.9	72.94	67.44	74.24	72.37
Gini	85.00	85.33	82.96	62.28	62.23	78.53	91.54	88.47
ATS	84.48	83.98	81.57	62.28	63.48	75.42	80.96	85.69
Vanilla Softmax	83.76	81.67	78.91	59.23	60.18	72.47	90.61	84.61
MC Dropout	83.54	80.89	77.91	59.23	61.46	65.71	80.39	83.09
PRIMA	84.55	83.1	79.36	60.08	58.96	73.82	91.22	85.73
RS	74.01	65.37	70.04	71.59	74.15	72.66	90.02	74.62
NSS	88.99	91.55	85.11	72.06	77.26	86.11	98.16	89.08

Table 9. Evaluation results of APFD for NSS and baselines.

Dataset(DNN)	NSS	NAC	KMNC	NPC	DSA	LSA	PRIMA	Softmax	Dropout	Gini	ATS	RS
MNIST (L-1)	99.18	45.73	41.86	45.27	90.68	88.43	89.25	98.22	97.43	98.23	98.48	53.27
MNIST (L-5)	98.84	43.87	39.85	42.95	84.92	80.56	92.53	97.49	97.58	98.17	98.29	49.83
Fashion (L-1	87.15	43.42	59.21	50.14	75.29	60.82	70.24	86.47	86.43	86.72	86.97	48.29
Fashion (R-20)	86.42	41.35	40.97	50.09	74.82	57.93	74.23	83.29	84.95	83.51	84.72	47.13
SVHN (L-5)	96.24	32.75	38.27	39.64	64.37	55.14	74.94	91.37	92.42	95.92	95.83	50.84
SVHN (V-16)	89.06	35.91	42.01	45.38	69.83	58.72	72.81	88.15	84.26	88.01	88.24	47.28
CIFAR-10 (V-16)	85.95	49.25	50.28	50.22	60.01	58.25	78.29	75.92	74.37	83.84	83.95	49.52
CIFAR-10 (R-20)	99.87	47.39	50.65	49.73	55.72	48.27	82.73	90.28	91.73	95.58	96.26	50.17

of uncertainty-guided baselines only obtains 84.55% (i.e., PRIMA [58]) while NSS obtains 88.99%. Finally, we can also observe that NSS obtains SOTA performance in other combinations in Tab. 8, which further illustrates NSS's fault detection effectiveness.

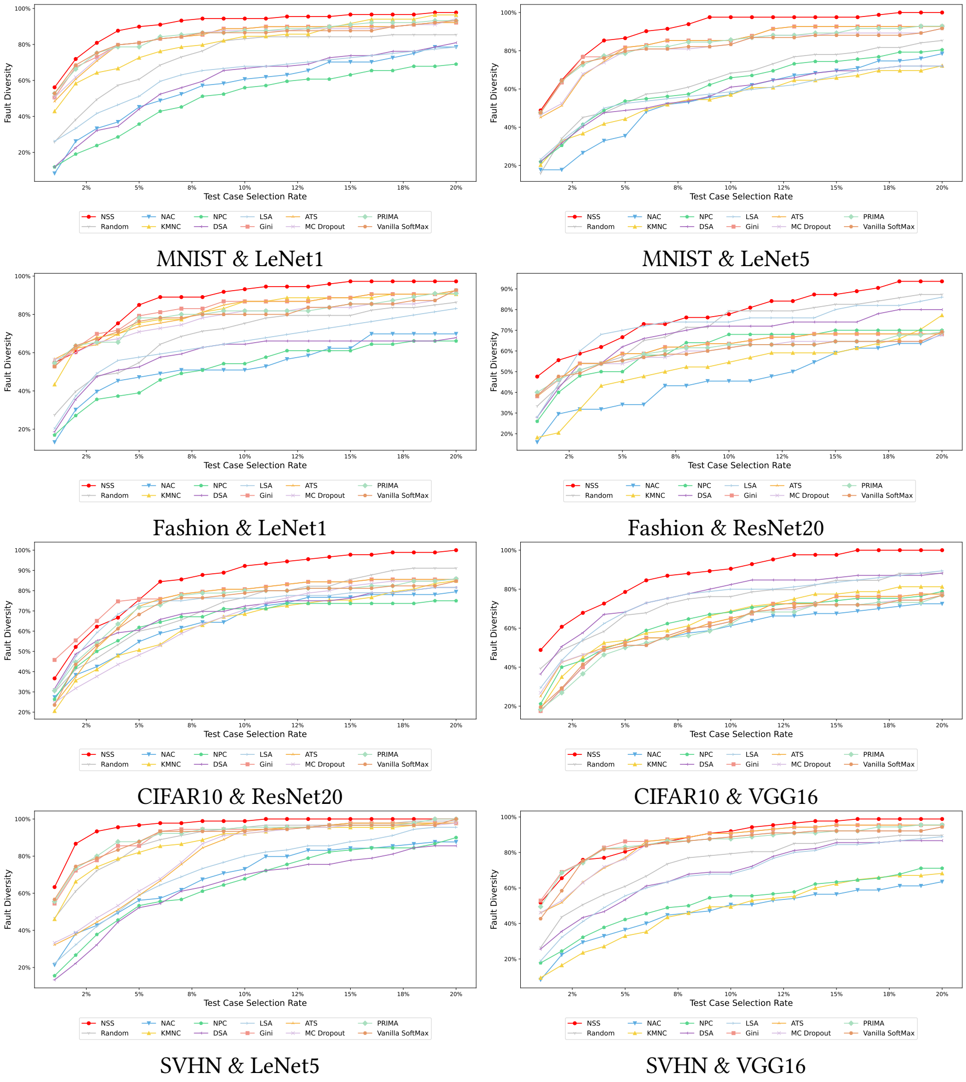
Answer to RQ2.2: NSS can detect more diverse error-inducing inputs/tests compared with baselines. For example, when we select 20% tests for MNIST&LeNet1 combination, NSS obtains 88.99% RAUC while baselines only obtain 85.00% RAUC.

4.3.3 *Average Percentage of Fault-Detection.*

In software testing, the Average Percentage of Fault-Detection (APFD) metric serves as a pivotal benchmark for assessing the efficiency of test case prioritization strategies. The APFD value is calculated using the formula:

$$APFD = 1 - \frac{\sum_{i=1}^{n} TF_i}{n \times m} + \frac{1}{2n}$$

where n represents the total number of test cases, m is the number of faults detected, and TF_i denotes the position of the first test case that reveals the ith fault, indexed from 1. This equation effectively measures the test suite's ability to



MNIST & LeNet1

MNIST & LeNet5

Fashion & LeNet1

Fashion & ResNet20

CIFAR10 & ResNet20

CIFAR10 & VGG16

SVHN & LeNet5

SVHN & VGG16

Fig. 5. The cumulative sum of the fault types coverage rate of our and baseline selection methods.

detect faults early, with a higher APFD value indicating superior test prioritization by allowing earlier detection of faults, thus optimizing testing efforts and resources.

To illustrate NSS's effectiveness in selecting valuable tests earlier, we report the APFD of NSS and baselines in Tab. 9, where we can observe that NSS obtains SOTA APFD results across various datasets and DNN architectures when prioritizing all tests in candidate datasets. Specifically, we can observe that compared with neuron-coverage-guided test case selection methods, NSS obtains higher RAUC in all combinations. For example, NSS obtains 99.18% RUAC in MNIST&LeNet1 combination, while NAC, KMNC, and NPC only obtain 45.73%, 41.86%, and 45.27% RAUC, which is even lower than RS that is consistent with previous study [12, 59]. Then, we can also observe that although surprise-adequacy-guided test case selection methods obtain higher RAUC compared with RS, they are still lower than NSS. For example, DSA and LSA obtain 90.68% and 88.43% in MNIST&LeNet1 combination, while MNIST&LeNet1 obtains 99.18% RAUC. Next, compared with uncertainty-guided methods, NSS still obtains SOTA RAUC in all combinations. For example, NSS improve the RAUC in MNIST&LeNet1 combination from 98.48% to 99.18%.

Answer to RQ2.3: NSS obtains higher APFD compared with baseline strategies. For example, when we prioritize all candidate tests for the MNIST&LeNet1 combination, NSS obtains 99.18% APFD while baselines only obtain 98.48% APFD at most.

4.3.4 Optimization Effectiveness. After selecting valuable tests from unlabeled candidate datasets, DNN developers will manually label the tests to retrain the DNN with these tests to improve DNN performance. To evaluate the effectiveness of NSS in optimizing the model, we collect the selected samples with four different ratios (5%, 10%, 15%, 20%) and add them to the original training set for further model tuning. The selection strategy employed is consistent with that mentioned in Tab. 7. For each dataset & model combination, we use the same training hyperparameters (e.g., epoch, optimizer settings) to perform a fair comparison. Specifically, all models are trained for 40 epochs, with the learning rate initialized to 0.001 and stepped down to one-tenth of the original in the 20th and 30th epochs. We use SGD as the optimizer with a Nesterov momentum of 0.99.

We provide the DNN accuracy improvement results in Tab. 10, where we can observe that NSS obtains SOTA performance in most of the experiments. Specifically, compared with neuron-coverage-guided methods, we can observe that NSS obtains SOTA performance in all experiments. For example, when we finetune DNNs with 5% tests which were selected from the candidate dataset, NSS increases 9.01% accuracy in the MNIST&LeNet1 dataset, while NAC, KMNC, and NPC only increased 7.07%, 7.31%, and 7.10% accuracy. As shown in Tab. 10, we can also observe that the accuracy improvement of DNNs fine-tuned with tests selected by neuron-coverage-guided methods is close to RS's result, which is consistent with previous study [12, 13, 56]. Then, compared with surprise-adequacy-guided methods, we can observe that NSS is also better than DSA and LSA for all experiments. For example, as shown in Tab. 10 when we finetune DNNs with 5% tests selected from candidate dataset for Fashion&LeNet1 combination, NSS improve 9.43% accuracy while DSA and LSA only improve 4.97% 5.17% accuracy. Finally, compared with uncertainty-guided methods, we can observe that in most experiments, NSS obtains SOTA performance, while only in MNIST&LeNet1 combination, DeepGini obtains SOTA performance compared with NSS. For example, when we finetune DNNs with 5% tests for MNIST&LeNet1 combination, DeepGini increases 7.57% accuracy, while NSS only obtains 7.44% accuracy. However, we can also notice that in other model and dataset combinations, NSS still has SOTA performances.

In conclusion, the empirical evaluation of NSS demonstrates its superior capability in enhancing DNN performance. This approach not only surpasses traditional neuron-coverage-guided methods in all test scenarios but also outperforms surprise-adequacy-guided and certain uncertainty-guided methods, showcasing its effectiveness in optimizing DNN accuracy. Despite the competitive performance of methods like DeepGini in specific cases, NSS consistently emerges as the optimal choice for DNN developers seeking to improve model performance through targeted test selection.

Table 10. Increase in DNN's accuracy (%) after repairing the DNN with test cases selected by DLS testing.

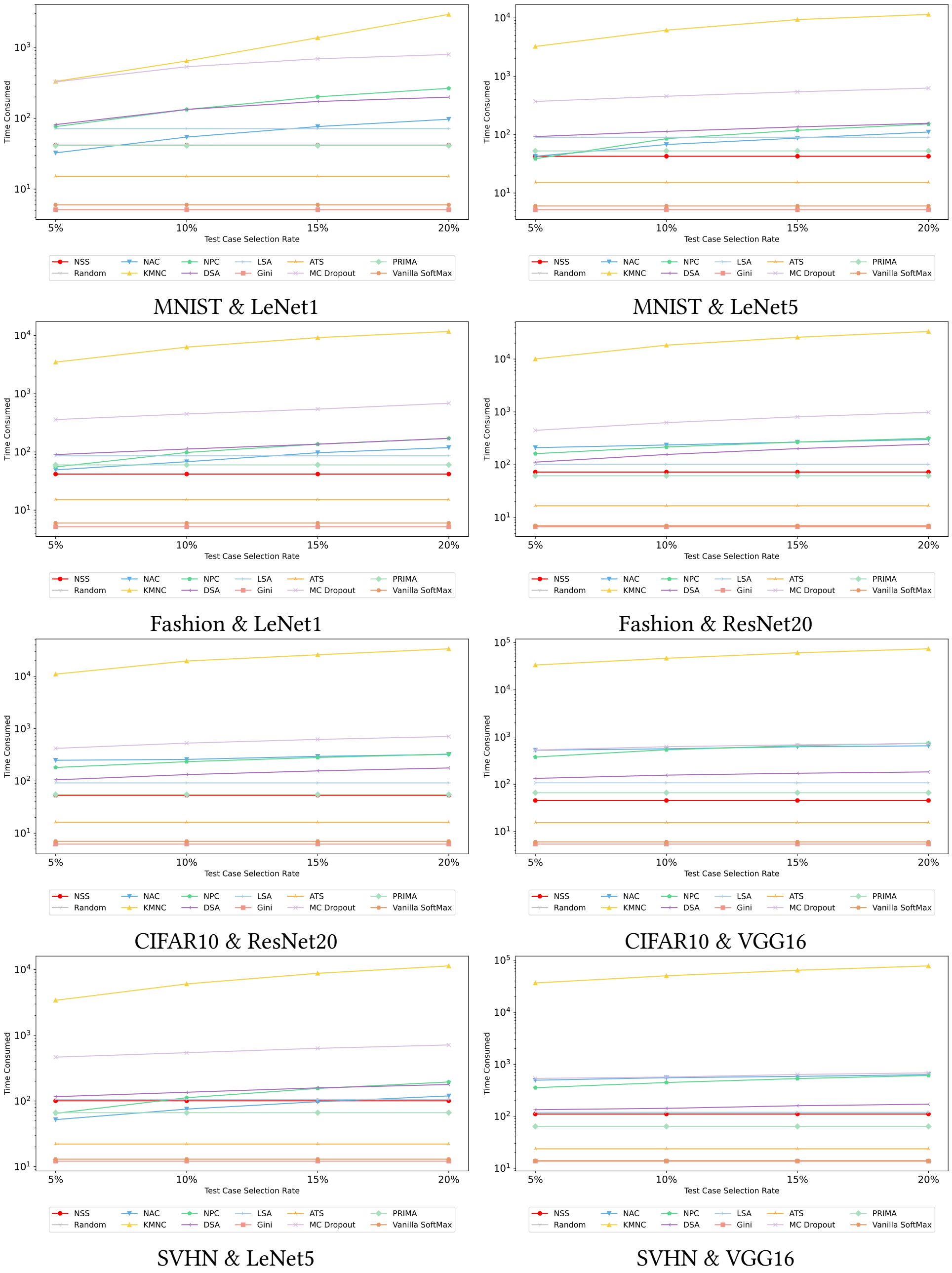
Dataset(DNN)	Select 5% Test Cases											
	NSS	NAC	KMNC	NPC	DSA	LSA	PRIMA	Softmax	Dropout	Gini	ATS	RS
MNIST (L-1)	9.01	7.07	7.31	7.10	7.10	7.39	7.57	7.38	7.62	7.54	7.41	7.09
MNIST (L-5)	7.44	7.04	7.24	7.03	7.04	7.33	7.12	7.43	7.51	7.57	7.21	7.04
Fashion (L-1)	9.43	4.95	5.01	4.98	4.97	5.17	5.74	6.12	5.85	5.31	5.73	4.97
Fashion (R-20)	6.79	6.08	6.10	6.09	6.10	6.08	6.12	6.52	6.37	6.13	6.15	6.08
SVHN (L-5)	5.77	3.86	3.94	3.85	3.86	4.25	4.73	4.23	4.31	4.21	4.23	3.85
SVHN (V-16)	3.35	2.11	2.41	2.12	2.12	2.32	3.34	2.59	2.47	2.57	2.81	2.11
CIFAR-10 (V-16)	2.86	1.51	1.61	1.50	1.51	1.74	1.76	2.07	2.11	2.12	2.05	1.50
CIFAR-10 (R-20)	4.62	2.35	2.47	2.35	2.34	2.51	2.83	2.88	2.74	2.79	2.81	2.35

Dataset(DNN)	Select 10% Test Cases											
	NSS	NAC	KMNC	NPC	DSA	LSA	PRIMA	Softmax	Dropout	Gini	ATS	RS
MNIST (L-1)	9.04	7.14	7.42	7.14	7.15	7.81	7.93	7.48	7.65	7.91	7.82	7.14
MNIST (L-5)	7.46	7.06	7.31	7.08	7.07	7.42	7.21	7.43	7.51	8.21	7.24	7.06
Fashion (L-1)	9.50	5.04	5.11	5.05	5.03	5.46	6.13	6.24	5.97	5.77	6.23	5.04
Fashion (R-20)	6.82	6.14	6.23	6.12	6.16	6.15	6.53	6.55	6.46	6.37	6.42	6.14
SVHN (L-5)	5.85	4.12	4.22	4.13	4.14	4.39	5.29	4.59	4.72	5.21	5.17	4.13
SVHN (V-16	3.56	2.40	2.51	2.38	2.39	2.57	3.47	2.83	2.70	2.92	3.03	2.38
CIFAR-10 (V-16)	3.33	1.82	1.91	1.81	1.82	1.92	2.59	2.35	2.42	2.73	2.33	1.81
CIFAR-10 (R-20)	4.73	2.44	2.54	2.43	2.41	2.55	3.02	3.14	2.98	3.02	3.15	2.43

Dataset(DNN)	Select 15% Test Cases											
	NSS	NAC	KMNC	NPC	DSA	LSA	PRIMA	Softmax	Dropout	Gini	ATS	RS
MNIST (L-1)	9.04	7.20	7.49	7.21	7.20	7.85	7.97	7.52	7.69	8.03	7.93	7.20
MNIST (L-5)	7.46	7.07	7.37	7.09	7.08	7.54	7.34	7.43	7.51	8.43	7.25	7.08
Fashion (L-1)	9.53	5.10	5.21	5.09	5.08	5.53	6.25	6.53	6.07	6.01	6.31	5.09
Fashion (R-20)	6.86	6.26	6.31	6.27	6.27	6.38	6.67	6.63	6.48	6.52	6.60	6.26
SVHN (L-5)	6.10	4.38	4.45	4.36	4.37	4.77	5.56	4.83	4.86	5.77	5.48	4.37
SVHN (V-16)	3.67	2.57	2.65	2.58	2.57	2.85	3.55	3.24	3.09	3.16	3.17	2.57
CIFAR-10 (V-16)	3.84	2.02	2.22	2.04	2.03	2.19	3.13	3.15	3.17	3.21	2.51	2.04
CIFAR-10 (R-20)	4.82	2.55	2.71	2.55	2.56	2.67	3.59	3.42	3.29	3.47	3.41	2.56

Dataset(DNN)	Select 20% Test Cases											
	NSS	NAC	KMNC	NPC	DSA	LSA	PRIMA	Softmax	Dropout	Gini	ATS	RS
MNIST (L-1	9.07	7.26	7.53	7.26	7.27	7.89	8.11	7.65	7.78	8.07	8.01	7.26
MNIST (L-5)	7.47	7.09	7.41	7.10	7.08	7.83	7.55	7.44	7.52	8.62	7.25	7.08
Fashion (L-1)	9.63	5.12	5.24	5.12	5.13	5.68	6.34	6.79	6.24	6.15	6.47	5.12
Fashion (R-20)	6.88	6.31	6.37	6.32	6.31	6.43	6.82	6.71	6.59	6.63	6.63	6.32
SVHN (L-5)	6.33	4.61	4.72	4.61	4.59	4.89	5.82	5.08	4.97	5.92	5.69	4.61
SVHN (V-16)	3.72	2.80	2.92	2.82	2.79	2.92	3.56	3.47	3.28	3.32	3.40	2.81
CIFAR-10 (V-16)	4.33	2.49	2.57	2.49	2.49	2.89	3.63	3.42	3.59	3.37	2.77	2.49
CIFAR-10 (R-20)	4.92	2.71	2.77	2.72	2.72	2.85	4.05	3.75	3.63	3.91	3.76	2.71

Answer to RQ2.4: NSS selected tests can improve more accuracy compared with baselines. For example, when we finetune DNN with 5% tests for MNIST&LeNet1 combination, NSS improves 9.01% accuracy while baselines only improve 7.57% accuracy.



MNIST & LeNet1

MNIST & LeNet5

Fashion & LeNet1

Fashion & ResNet20

CIFAR10 & ResNet20

CIFAR10 & VGG16

SVHN & LeNet5

SVHN & VGG16

Fig. 6. Overhead on all dataset&model combinations. The vertical axis is the logarithmic coordinate.

4.3.5 Overhead. Test case selection is used to select valuable test cases from a large number of unlabeled candidate datasets, which can reduce the labeling efforts of DNN developers. However, if the selection method has a high overhead, it can negatively impact the efficiency and practicality of the overall testing process. To evaluate the overhead of NSS and the baselines, we record the time consumed by each selection method when the proportion of selected samples among all candidates is 5%, 10%, 15%, and 20%, respectively. In this experiment, we take the time consumption for random selection as 0 seconds. The evaluation results are shown in Fig. 6. First, we can observe that neuron-coverage-guided selection methods (i.e., NAC, KMNC, and NPC) approximately consume hundreds of seconds on smaller models and close to or over 10^3 seconds on larger models. The slowest KMNC-based selection generally takes several hours to complete, consistent with recent research results [12, 13], which is due to neuron-coverage metric guided selection methods are based on a greedy search to select test samples one by one with time complexity $O(mn^2)$ [12], where m is the number of elements (e.g., neurons) to cover and n is the size of unlabeled datasets. Then, we can also observe that NSS is more efficient compared with surprise-adequacy-guided test case selection methods in most experiments. However, for the SVHN dataset, LSA obtains competitive results with NSS, which is due to the SVHN dataset having more data samples in the dataset compared with other datasets. Then, NSS will require more time for `ArgSort` function. Finally, we can also observe that NSS will require more time compared with uncertainty-guided selection methods. For example, NSS is slower than DeepGini in all experiments which is because DeepGini only requires calculating the Gini score and the use `ArgSort` function to select tests with higher Gini scores. However, NSS will first utilize *Sensitive Neuron Identifier* to detect sensitive neurons in DNNs and then calculate TNSScore for these neurons, which inevitably requires more time for NSS. We also notice that ATS will spend more time compared with DeepGini and NSS, which is because ATS will calculate the extra pattern and fitness score, which cause it slower than NSS and Gini. We can also observe in uncertainty-guided selection methods, that only MC Dropout requires more time compared with other methods (i.e., requires 10^2 to 10^3 seconds for test case selection), which is due to MC Dropout requires several times dropout process since the sample parameters are 200 (default setting in other papers [59].).

Answer to RQ2.5: NSS obtains competitive results compared with uncertainty-guided test case selection methods.

4.3.6 Breaking down the overhead of NSS. As shown in Fig. 6, we illustrate the overhead of NSS and baseline strategies. In this section, we further break down the overhead of NSS in Tab. 11. Specifically, as illustrated in Alg. 1 and Alg. 2, two main components contribute to the overhead of NSS: Sensitive Neuron Identifier and Test Case Selection. During the selection process, NSS will first use a small subset in the dataset, where both the Sensitive Neuron Identifier and Test Case Selection use BenignMutation to obtain mutation cases for the input x. Then x and x' are fed into the models and the neuron sensitivity is calculated. However, the test case selection will only compute the neuron sensitivity for the sensitive neurons for all candidate datasets. As shown in Tab. 11, we can observe that when we select 20% tests based on the TNSScore for sensitive neurons (i.e., we first utilize a small subset to detect sensitive neurons in DNNs and then calculate TNSScore in these neurons), we only require about 10% time compared with selecting 20% tests based on all neurons' TNSScore in DNNs since when we first identify sensitive neurons in DNNs, the test case selection process only requires a few times (e.g. about 10.06% to 18.51% in Tab. 11) compared to the Sensitive Neuron Identifier.

Answer to RQ2.6: Sensitive Neuron Identifier largely reduce the overhead of NSS. For example, with the Sensitive Neuron Identifier, the overhead of MNIST&LeNet1 decreases from 382.74s to 42.38s.

Table 11. Time overhead breakdown for NSS across different datasets and models when we select 20% test cases from candidate dataset. Full Result means that we select 20% test cases based on all neuron's TNSScore.

Dataset	MNIST		Fashion		CIFAR10		SVHN	
Model	LeNet1	LeNet5	LeNet1	ResNet20	VGG16	ResNet20	LeNet5	VGG16
Test Case Selection	5.62	5.68	5.76	6.68	7.40	7.18	12.61	15.67
Sensitive Neuron Identifier	36.76	37.13	36.77	66.40	39.97	46.60	89.36	95.92
Full Result	382.74	412.52	371.59	682.73	415.28	489.33	815.72	1035.46

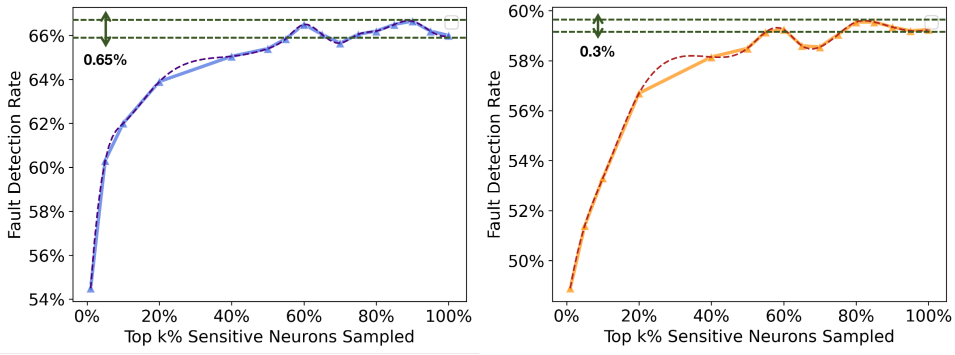
Table 12. Fault detection rate with the ratio of sensitive neurons (sampled from the last encoder layer) ranging from 1% to 100%. We report the FDR for 20% of the test cases.

Dataset (DNN)	Top k% sensitive neurons				
	1%	5%	10%	20%	100%
MNIST (LeNet-1)	48.9%	51.4%	53.3%	56.7%	59.3%
MNIST (LeNet-5)	44.8%	50.0%	50.9%	51.8%	54.3%
Fashion (LeNet-1)	54.4%	57.0%	57.4%	61.2%	58.3%
Fashion (ResNet-20)	54.5%	60.3%	62.0%	63.9%	66.0%
SVHN (LeNet-5)	47.1%	50.9%	51.2%	51.8%	52.2%
SVHN (VGG-16)	41.4%	43.1%	43.3%	45.7%	46.3%
CIFAR-10 (VGG-16)	55.7%	56.6%	60.7%	60.0%	60.9%
CIFAR-10 (ResNet-20)	52.9%	53.9%	54.1%	54.9%	55.1%

4.4 How does sensitive neuron size affect NSS's effectiveness?

The FDR experiment results in Tab. 7 are based on Top-10% most sensitive neurons across all datasets and models, which derives from our empirical studies where we can obtain adequate performance (FDR) while maintaining sufficient selection efficiency with Top-10% neurons. However, a fixed percentage of neurons may not generalize well to models, so we also study how the proportion of sensitive neurons tested would affect the estimation of the *value* of a test case and further affect the FDR.

The evaluation results are illustrated in Tab. 12, where we can observe that first when we increase the percentage of sensitive neurons in DNNs, the FDR will consistently increase in most experiments. For example, when we increase the $Topk\%$ from 1% to 100%, the FDR of NSS will increase from 48.9% to 59.3% for MNIST&LeNet1 combination, which is consistent with our intuition, as more neurons tend to yield more accurate estimates. However, as discussed in Sec. 4.3.6, when we increase the $TopK\%$, the overhead of NSS will further increase, which indicates that there is a trade-off between the FDR and the overhead of the NSS testing process. Second, we can also observe that the FDR will decrease when we increase the $Topk\%$ to 100% for some combinations. For example, when we increase the $Topk\%$ from 20% to 100%, the FDR will decrease from 61.2% to 58.3% for the Fashion&LeNet1 combination. We can observe that in Fig. 7, similar behaviors also existed in the Fashion&ResNet20 and MNIST&LeNet1 combinations. For example, when we increase the $Topk\%$ from 60% to 100% the FDR will decrease 0.65% for the Fashion&ResNet20 combination. We can also observe that MNIST&LeNet1 will also decrease 0.3% FDR when the $Topk\%$ increases to 100% compared with its optimal performance, which indicates that directly utilizing all internal neurons for test case selection may not obtain SOTA FDR for test case selection process.



Fashion & ResNet20

MNIST & LeNet1

Fig. 7. The correlation between FDR and the number of sampled neurons. The dashed line represents the data obtained by cubic spline interpolation.

Table 13. NSS's fault detection rate (%) with different model layers where sensitive neurons are sampled from. Layer -1 and Layer -2 refer to the final and penultimate layers of the network's encoder component, respectively. While Layer 1 and Layer 2 refer to the first and the second layers of the network's encoder component. Gini-Layer is the last layer in the DNNs that was previously used to calculate Gini values.

Dataset (DNN)	Sensitive neuron selection layer				
	Layer1	Layer2	Layer -2	Layer -1	Gini-Layer
MNIST (LeNet-1)	56.65	56.50	56.70	60.05	56.71
MNIST (LeNet-5)	51.65	51.50	51.80	53.55	51.35
Fashion (LeNet-1)	57.15	57.00	61.20	60.40	58.32
Fashion (ResNet-20)	44.25	46.30	60.65	63.90	49.37
SVHN (LeNet-5)	47.69	51.65	51.79	56.25	50.28
SVHN (VGG-16)	21.17	21.65	42.04	45.76	53.75
CIFAR-10 (VGG-16)	40.45	40.25	52.40	60.00	45.71
CIFAR-10 (ResNet-20)	30.25	33.60	40.00	54.85	48.92

Answer to RQ3: The ratio of sensitive neurons affects the result of NSS. In most of the experiments, increasing the ratio of sensitive neurons will increase the FDR of NSS, but it will also increase the testing overhead.

4.5 How does the selected layer affect NSS's effectiveness?

Our evaluation results in Tab. 7 directly use the last encoder layer in DNNs to evaluate NSS's fault detection effectiveness, which inspired us to discuss whether the layer used in NSS will affect its fault detection effectiveness. To answer this question, we evaluate NSS for different encoder layers in DNNs in Tab. 13, where we also evaluate NSS in the final layer which is commonly used by current uncertainty-guided selection methods (e.g., DeepGini [12] and ATS [13]). We can observe that first, in most of the experiments, NSS's effectiveness will increase when we consider a deeper encoder layer in DNNs. For example, when we choose the first encoder layer in the CIFAR10&ResNet-20 combination, the FDR of NSS only has 30.25%, while when we choose Layer -2 and Layer -1, the FDR will increase to 40.00% and 54.85%, which indicate that when we utilize deeper encoder layer in DNNs for NSS to select tests from candidate dataset, the FDR

Table 14. Fault detection rate (FDR) of NSS with different mutation strategies for Alg. 1 line 5-6. Experiments are conducted on the CIFAR-10 dataset with ResNet20 and VGG16 models.

	ResNet20				VGG16			
Mutation	5%	10%	15%	20%	5%	10%	15%	20%
Shift	72.0%	63.4%	57.1%	52.6%	75.9%	67.8%	65.4%	62.8%
Rotation	69.8%	62.9%	58.2%	52.7%	73.4%	65.9%	62.7%	59.4%
Scale	70.2%	62.9%	57.4%	53.0%	68.4%	64.5%	62.7%	58.9%
Shear	71.4%	63.4%	58.3%	53.9%	69.2%	65.3%	63.1%	61.0%
Contrast	75.6%	65.1%	58.9%	53.9%	61.8%	57.6%	53.5%	50.8%
Brightness	76.0%	65.8%	60.5%	55.8%	64.2%	60.7%	55.9%	52.2%
Blur	75.6%	67.0%	60.6%	55.2%	67.3%	62.9%	59.8%	54.3%
all	71.4%	62.8%	56.7%	54.1%	68.6%	64.8%	63.4%	60.7%

of NSS will be higher. This pattern suggests that deeper encoder layers, which encapsulate a broader perceptual field and richer global semantic information, are more effective for fault detection with NSS. In contrast, shallower layers primarily capture local, low-level data features, limiting the scope of fault-related information that can be detected. The enhanced performance observed with deeper layers underscores the inherent architectural advantages of DNNs, where sensitive neurons in these layers are more likely to identify global fault-related information. This observation aligns with strategies employed in current backdoor trigger mitigation efforts, which typically analyze the last encoder layer to identify malicious modifications within DNNs [21].

Next, we also evaluate NSS in the DNN final layer (we note it as Gini-Layer) which is commonly used to calculate Gini values for DeepGini in Tab. 13. We can observe that the FDR of NSS in the Gini-Layer is lower than the last encoder layer of the encoder component in most of the experiments. For example, NSS obtains 60.05% FDR in the last encoder layer in MNIST&LeNet1 combination, while NSS only obtains 56.71% FDR in the same configuration. The reason may be the intrinsic difference in the type of information processed by these layers. The final fully connected layer (Gini-Layer) in DNNs, which is typically used to calculate the Gini index in prioritization techniques like DeepGini, is designed to make the final decision by integrating all the information processed in the previous layers. This layer emphasizes the overall confidence level of the network's prediction across different classes. On the other hand, the last encoder layer of the encoder component, which we focus on in our approach, processes more detailed and specific features that are crucial for the internal decision-making process of the DNN. These features include but are not limited to, spatial relationships, texture details, and other high-level attributes that are significant for recognizing patterns in the data.

Answer to RQ4: The selected layer will affect the effectiveness of NSS. Our evaluation results illustrate that NSS will have the SOTA FDR when we utilize the last encoder layer in our experiments.

4.6 Does different BenignMutation strategies affect NSS's effectiveness?

As shown in Alg. 1 and Alg. 2, NSS will utilize BenignMutation to identify sensitive neurons in the DNNs and select test cases from candidate datasets. To mutate test cases in candidate datasets, we utilize seven different benign mutations recommended by recent study [13, 54] to detect sensitive neurons in Tab. 7. In this section, we further evaluate NSS's effectiveness with different benign mutations in sensitive neuron identifier and test case selection.

Table 15. Fault detection rate (FDR) of NSS with different mutation strategies for Alg. 2 line 8-9. Experiments are conducted on the CIFAR-10 dataset with ResNet20 and VGG16 models.

	ResNet20				VGG16			
Mutation	5%	10%	15%	20%	5%	10%	15%	20%
Shift	52.0%	47.8%	44.9%	41.6%	79.0%	74.8%	67.0%	65.2%
Rotation	44.8%	40.1%	39.1%	37.2%	65.2%	60.3%	57.6%	54.6%
Scale	64.4%	57.8%	54.1%	50.2%	74.2%	67.5%	61.8%	56.6%
Shear	51.2%	49.8%	46.2%	43.4%	69.4%	65.7%	61.8%	58.6%
Contrast	48.4%	41.2%	38.5%	35.2%	63.0%	58.4%	53.2%	48.5%
Brightness	65.2%	56.1%	51.7%	48.6%	62.5%	56.3%	52.9%	48.7%
Blur	81.4%	80.6%	78.9%	77.0%	66.8%	62.4%	55.8%	49.4%
all	71.4%	62.8%	56.7%	54.1%	68.6%	64.8%	63.4%	60.7%

BenignMutation in Sensitive Neuron Identifier. We first provide the evaluation results of NSS with different mutation strategies in Sensitive Neuron Identifier. The evaluation results are shown in Tab. 14, where we can observe that first NSS obtains competitive performance in all mutation strategies. Specifically, NSS obtains 69.8% to 76.0% FDR in seven different mutation strategies when NSS selects 5% tests from the candidate dataset for CIFAR10&ResNet20 combination, which is close to the results that were selected based on all seven mutation strategies (i.e., 71.4%). Furthermore, we can also observe that NSS obtains state-of-the-art (SOTA) performance for the CIFAR10&ResNet20 combination with the brightness mutator and for the CIFAR10&VGG16 combination with the shift mutator. This indicates that certain mutators may be more beneficial for sensitive neuron identification.

BenignMutation in test case selection. The evaluation results are shown in Tab. 15, where we can observe that NSS's FDR will largely change with different mutation strategies to guide test case selection. For example, when we select 5% test cases from the candidate dataset for CIFAR10&ResNet20 combination, the FDR will change from 44.8% to 81.4% for seven different types of mutation strategies. We can also observe that NSS with Blur mutation obtains the highest FDR compared with other mutation strategies when we select 5% to 20% test cases (e.g., 81.4% to 77.0% FDR) from the candidate dataset for CIFAR10&ResNet20 combination. However, for the CIFAR10&VGG16 combination, the Shift mutation strategy consistently achieves the highest FDR across all selection percentages, ranging from 79.0% to 65.2%. This suggests that the effectiveness of different mutation strategies for test case selection may vary depending on the specific dataset and model combination.

Answer to RQ5: The choice of benign mutation strategy has a significant impact on the effectiveness of NSS. For the CIFAR10&ResNet20 combination, NSS achieves SOTA performance when using the Blur mutation strategy, with an FDR ranging from 81.4% to 77.0% when selecting 5% to 20% of the test cases from the candidate dataset. In contrast, for the CIFAR10&VGG16 combination, the Shift mutation strategy consistently yields the highest FDR, ranging from 79.0% to 65.2% across all selection percentages, which illustrate that the Blur mutator does not always obtain SOTA performance across all dataset and model combinations. In some cases, other mutators may outperform the Blur mutator. Since it is challenging to predetermine which mutator will achieve SOTA performance for the NSS testing process, our paper focuses on discussing the combination of seven mutators to provide a more comprehensive and robust approach.

4.7 Threats to Validity

The internal validity of NSS lies in our implementations, including BenignMutation generation, Sensitive Neuron Identifier, Test Case Selection algorithm in our code, and implementation in each experiment in our paper. To reduce the threat, four authors carefully checked the correctness of our implementation.

The external validity of NSS lies in the test subject selection and the tools used in the evaluation. First, the selection of datasets and DNN models could be an external threat. To reduce the threat, we follow the current study to select four widely used datasets (i.e., MNIST, SVHN, CIFAR10, and FashionMNIST) and four well-known pre-trained DNN models (i.e., LeNet1, LeNet5, VGG16, and ResNet20). Another threat could be the generated data including mutation samples in sensitive neuron identifier and test case selection. To mitigate this threat, we follow the same setting in the existing work [13, 54] with multiple seed configurations.

The construct validity of NSS mainly lies on the randomness, the selected baselines, metrics, and the parameters used by NSS and baselines. First, Alg. 1 and Alg. 2 will randomly select mutation strategies for each test case. To mitigate this randomness, we repeat all experiments 5 times and calculate the average results. For baseline selections, following recent study, we consider three types of test case selection methods [13, 59] (i.e., neuron-coverage-guided, surprise-adequacy-guided, and uncertainty-guided test case selection). For each type of selection method, we adopted state-of-the-art test case selection methods that have been widely evaluated by our baseline strategies. We compare NSS with these SOTA baselines to demonstrate the effectiveness of our proposed test case selection. Next, the metrics used in our paper could be a threat. To reduce the threat, we follow recent studies [12, 13, 59] to utilize FDR, RAUC, and APFD to measure fault detection effectiveness of NSS and baselines. Finally, the parameters used in our method and baselines could be a threat. To reduce the threat, for baselines, we follow the setting of existing works [12, 13, 44, 59, 63]. For our method, we set multiple different configurations for $TopK\%$.

5 RELATED WORK

This section discusses related work in three groups: neuron-coverage-guided test case selection, surprise-adequacy-guided test case selection, and uncertainty-guided test case selection.

5.1 Neuron-coverage-guided Test Case Selection

Pei et al. [44] propose the first white-box coverage criteria (i.e., NAC), which calculates the percentage of activated neurons. DeepGauge [36] then extends NAC and proposes a set of more fine-grained coverage criteria (e.g., KMNC, NBC, SNAC, and TKNC) by considering the distribution of neuron outputs from the training data. Inspired by the coverage criteria in traditional software testing, some coverage metrics [35, 37, 51] are proposed. DeepCover [51] proposes the MC/DC coverage of DNNs based on the dependence between neurons in adjacent layers. To explore how inputs affect neuron internal decision logic flow, Xie et al. [63] propose NPC to explore the neuron path coverage. DeepCT [35] adopts the combinatorial testing idea and proposes a coverage metric that considers the combination of different neurons at each layer. DeepConcolic [52] analyzed the limitation of existing coverage criteria and proposed a more fine-grained coverage metric that considers the relationships between two adjacent layers and the combinations of values of neurons at each layer. Recently, Liu et al. [34] proposed DeepState to select data based on a stateful perspective of RNN, which identifies the possibly misclassified test by capturing the state changes of neurons in RNN models. Since DeepState needs to capture the LSTM and other similar RNNs to analyze its state change, so DeepState is mainly used in RNN. DeepImportance [14] proposes the concepts of *important neuron*, which are core contributors in decision-making, by only testing *important neuron*'s coverage can decrease the testing time. Our study demonstrated that using these coverage criteria, coverage-based test prioritization is not effective and efficient (Sec. 4.3), which is usually time-consuming and highly expensive. Sometimes, its effectiveness is even worse than random prioritization. Instead, our approach uses a simple metric by analyzing internal neuron sensitivity to become more effective and efficient.

5.2 Surprise-Adequacy-guided Test Case Selection

The surprise-adequacy-guided test case selection method identifies test cases that are most surprising when compared to the training set. As the pioneering method of surprise adequacy for deep learning testing, Kim et al. [22] introduced Likelihood-based Surprise Adequacy (LSA) and Distance-based Surprise Adequacy (DSA). LSA calculates the surprise of a test input by estimating the negative log-likelihood of its activation traces using a Gaussian Kernel Density Estimator, which is parameterized based on the training set. Conversely, DSA measures surprise as the ratio of the Euclidean distance between a test input's activation traces and the nearest training activation traces of the same class, to the distance between that point and the nearest training activation traces of a different class. To reduce the overhead of DSA, Kim et al. [23] propose Mahalanobis-Distance Based SA (MDSA), which offers an efficient alternative to likelihood-based SA by calculating the likelihood of a test's activation traces using only the covariance matrix of the training set, enabling constant prediction runtimes and potentially reduced memory requirements through online covariance estimation for large datasets. However, recent studies [23, 59] also mentioned that although MDSA improves the efficiency of the test case selection process, the FDR and APFD of MDSA were lower than DSA. Recently, Kim and Yoo [24] proposed Multi-Modal surprise-adequacy-guided test case selection, introducing Multimodal LSA (MLSA) and Multimodal MDSA (MMDSA) to address the complexities of multi-modal distributions in the training set. MLSA utilizes a Gaussian Mixture Model (GMM) for a more accurate estimation of test input surprise by accounting for the diverse distributions of activation traces, whereas MMDSA clusters activation traces with the k-means algorithm, applying MDSA within each cluster to tailor the assessment of surprise to the nuanced structure of the data. Although surprise-adequacy-guided methods have been widely used for test case selection Feng et al. [12], Gao et al. [13], Kim et al. [22], Wang et al. [58], Weiss and Tonella [59], Xie et al. [63], researchers also mentioned that they are less effective compared with uncertainty-guided test case selection methods.

5.3 Uncertainty-guided Test Case Selection

The uncertainty-guided test case selection aims to select test cases in DNNs that exhibit low confidence in their predictions, thereby highlighting potential vulnerabilities. As the first uncertainty-guided test case selection for DNNs, DeepGini [12] computes a Gini score for inputs based on the output of the final layer, serving as a heuristic for uncertainty. Building on this foundation, Li et al. [32] introduced CES, a method that prioritizes test cases using Cross Entropy-based Sampling derived from the final layer's output. Besides, Gao et al. [13] proposed ATS, which leverages the difference between the model output to measure the behavior diversity of the DNN test case. However, since ATS requires the calculation of fitness scores for all classes, it requires more computation times compared with DeepGini. Furthermore, Wang et al. [58] developed PRIMA, a pioneering approach that prioritizes test inputs capable of eliciting divergent predictions across various mutated model versions[†]. Weiss and Tonella [59] conducted an empirical analysis demonstrating that straightforward uncertainty-guided techniques such as employing the Vanilla Softmax output,

[†] After discussing with Wang et al., we classify PRIMA as an uncertainty-guided method.

Softmax Entropy, and Monte-Carlo Dropout (MC Dropout) can achieve state-of-the-art (SOTA) performance in test case selection. Additionally, Weiss and Tonella [59] proposed the Prediction-Confidence Score (PCS) guided test case selection, which selects tests by examining the softmax likelihood disparity between the predicted class and its closest competitor. Our observation is that uncertainty-guided methods are focused on the model's final layer's output and ignore utilizing internal neuron information, which means that these uncertainty-guided methods may ignore the internal-neuron-related test cases which will induce internal neurons into an error and then cause the model to have incorrect behaviors even though the final layer output has high confidence. For example, ANP [61] and FMP [21] mentioned that some test cases can cause internal neurons to have incorrect behaviors while the final layer output has high confidence. Our method utilizes internal neuron information by proposing the test case's neuron sensitivity score to reveal how a test case affects internal neuron behaviors, which causes our method to become more effective and efficient.

6 CONCLUSION

In this paper, we propose NSS, a neuron sensitivity-guided test case selection method, and we also propose a sensitive neuron identifier to detect sensitive neurons in the model. The experimental results of the paper show that NSS can efficiently find more valuable test cases, which can be used to improve the quality of the model. NSS combines the advantages of existing neuron coverage criteria and prioritization techniques (i.e., allows the selected test cases to find neuron's corner cases), and these test cases also have higher confidence triggering the model into error. Our evaluation results illustrate that utilizing NSS selected tests to fine-tune DNN models can improve accuracy more than baselines. For example, when we select 5% test cases from the MNIST & LeNet1 combination, NSS obtains 81.8% FDR, when we use these test cases to finetune the LeNet1, it can increase 9.01% accuracy.

7 ACKNOWLEDGEMENTS

The work is supported in part by National Key R&D Program of China (2022ZD0160201), National Key R&D Program of China (No.2023YFB4503902), HK RIF (R7030-22), HK ITF (GHP/169/20SZ), the Huawei Flagship Research Grants in 2021 and 2023, and HK RGC GRF (Ref: HKU 17208223), the HKU-SCF FinTech AcademyR&D Funding Schemes in 2021 and 2022, the HKU-CAS Joint Laboratory for Intelligent System Software, and the Shanghai Artificial Intelligence Laboratory. This work is also supported by the National Research Foundation, Singapore, and the Cyber Security Agency under its National Cybersecurity R&D Programme (NCRP25-P04-TAICeN). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and Cyber Security Agency of Singapore. This work is also supported by the National Natural Science Foundation of China Grant No. 62322208.

REFERENCES

[1] 2022. *2021 Disengagement Report from California*. https://thelastdriverlicenseholder.com/2022/02/09/2021-disengagement-report-from-california/

[2] Raja Ben Abdessalem, Shiva Nejati, Lionel Claude Briand, and Thomas Stifter. 2018. Testing Vision-Based Control Systems Using Learnable Evolutionary Algorithms. *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)* (2018), 1016–1026. https://api.semanticscholar.org/CorpusID:47017345

[3] Jimmy Ba and Rich Caruana. 2013. Do Deep Nets Really Need to be Deep?. In *NIPS*.

[4] Mihalj Bakator and Dragica Radosav. 2018. Deep Learning and Medical Diagnosis: A Review of Literature. *Multimodal Technologies and Interaction* (2018).

[5] Christian Birchler, Sajad Khatiri, Bill Bosshard, Alessio Gambi, and Sebastiano Panichella. 2021. Machine learning-based test selection for simulation-based testing of self-driving cars software. *Empirical Software Engineering* 28 (2021). https://api.semanticscholar.org/CorpusID:243847320

[6] Mariusz Bojarski, David W. del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. 2016. End to End Learning for Self-Driving Cars. *ArXiv* abs/1604.07316 (2016).

[7] 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 (TOSEM)* 29 (2020), 1 – 35.

[8] Tsong Yueh Chen, Hing Leung, and I. K. Mak. 2004. Adaptive Random Testing. *2008 The Eighth International Conference on Quality Software* (2004), 443–443. https://api.semanticscholar.org/CorpusID:8225146

[9] Tsong Yueh Chen, Robert G. Merkel, Gary Eddy, and P. K. Wong. 2004. Adaptive random testing through dynamic partitioning. *Fourth International Conference onQuality Software, 2004. QSIC 2004. Proceedings.* (2004), 79–86. https://api.semanticscholar.org/CorpusID:29180953

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[11] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

[12] 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* (Virtual Event, USA) *(ISSTA 2020)*. Association for Computing Machinery, New York, NY, USA, 177–188. https://doi.org/10.1145/3395363.3397357

[13] Xinyu Gao, Yang Feng, Yining Yin, Zixi Liu, Zhenyu Chen, and Baowen Xu. 2022. Adaptive Test Selection for Deep Neural Networks. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. 73–85. https://doi.org/1O.1145/3510003.3510232

[14] Simos Gerasimou, Hasan Ferit Eniser, Alper Sen, and Alper Cakan. 2020. Importance-Driven Deep Learning System Testing. *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)* (2020), 322–323.

[15] Milo Gligorić, Lamyaa Eloussi, and Darko Marinov. 2015. Practical regression test selection with dynamic file dependencies. *Proceedings of the 2015 International Symposium on Software Testing and Analysis* (2015).

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

[17] 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? *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 851–862. https://doi.org/10.1145/3368089.3409754

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

[19] 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 (TOSEM)* 31, 4 (2022), 1–30.

[20] Qiang Hu, Lei Ma, Xiaofei Xie, Bing Yu, Yang Liu, and Jianjun Zhao. 2019. DeepMutation++: A Mutation Testing Framework for Deep Learning Systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1158–1161. https://doi.org/10.1109/ASE.2019.00126

[21] Dong Huang, Qi Bu, Yahao Qing, Yichao Fu, and Heming Cui. 2024. FMP: Adversarial Feature Map Pruning for Backdoor. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=IOEEDkla96

[22] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding deep learning system testing using surprise adequacy. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1039–1049.

[23] 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), 1–29.

[24] Seah Kim and Shin Yoo. 2021. Multimodal surprise adequacy analysis of inputs for natural language processing DNN models. In *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*. IEEE, 80–89.

[25] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report.

[26] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. 2018. Adversarial examples in the physical world. In *Artificial intelligence safety and security*. Chapman and Hall/CRC, 99–112.

[27] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. 2009. Exploring Strategies for Training Deep Neural Networks. *Journal of Machine Learning Research* 10, 1 (2009), 1–40. http://jmlr.org/papers/v10/larochelle09a.html

[28] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[29] 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* (Virtual Event, USA) *(ISSTA 2020)*. Association for Computing Machinery, New York, NY, USA, 165–176. https://doi.org/10.1145/3395363.3397346

[30] Owolabi Legunsen, Farah Hariri, August Shi, Yafeng Lu, Lingming Zhang, and Darko Marinov. 2016. An extensive study of static regression test selection in modern software evolution. *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (2016).

[31] Zenan Li, Xiaoxing Ma, Chang Xu, and Chun Cao. 2019. Structural coverage criteria for neural networks could be misleading. In *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 89–92.

[32] Zenan Li, Xiaoxing Ma, Chang Xu, Chun Cao, Jingwei Xu, and Jian Lü. 2019. Boosting Operational DNN Testing Efficiency through Conditioning *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 499–509. https://doi.org/10.1145/3338906.3338930

[33] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and X. Zhang. 2018. Trojaning Attack on Neural Networks. In *Network and Distributed System Security Symposium*.

[34] Zixi Liu, Yang Feng, Yining Yin, and Zhenyu Chen. 2022. DeepState: selecting test suites to enhance the robustness of recurrent neural networks. In *Proceedings of the 44th International Conference on Software Engineering*. 598–609.

[35] L. Ma, Felix Juefei-Xu, Minhui Xue, Bo Li, Li Li, Yang Liu, and Jianjun Zhao. 2019. DeepCT: Tomographic Combinatorial Testing for Deep Learning Systems. *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (2019), 614–618.

[36] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. 2018. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 120–131.

[37] L. Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. 2018. DeepMutation: Mutation Testing of Deep Learning Systems. *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)* (2018), 100–111.

[38] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* (2017).

[39] Agnieszka Mikołajczyk and Michał Grochowski. 2018. Data augmentation for improving deep learning in image classification problem. In *2018 international interdisciplinary PhD workshop (IIPhDW)*. IEEE, 117–122.

[40] Géraldin Nanfack, A Fulleringer, Jonathan Marty, Michael Eickenberg, and Eugene Belilovsky. 2023. Adversarial Attacks on the Interpretation of Neuron Activation Maximization. *ArXiv* abs/2306.07397 (2023). https://api.semanticscholar.org/CorpusID:259145377

[41] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. 2011. Reading Digits in Natural Images with Unsupervised Feature Learning. (2011).

[42] Anh Nguyen, Jason Yosinski, and Jeff Clune. 2019. Understanding Neural Networks via Feature Visualization: A survey. *ArXiv* abs/1904.08939 (2019).

[43] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 372–387.

[44] 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*. 1–18.

[45] José Luís Rivas Pizarroso, Javier Portela, and Antonio Muñoz. 2020. NeuralSens: Sensitivity Analysis of Neural Networks. *J. Stat. Softw.* 102 (2020).

[46] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. 2020. A Survey of Deep Active Learning. *ACM Computing Surveys (CSUR)* 54 (2020), 1 – 40.

[47] Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of big data* 6, 1 (2019), 1–48.

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

[49] Jeongju Sohn, Sungmin Kang, and Shin Yoo. 2019. Search based repair of deep neural networks. *arXiv preprint arXiv:1912.12463* (2019).

[50] Felix Stahlberg. 2020. Neural machine translation: A review. *Journal of Artificial Intelligence Research* 69 (2020), 343–418.

[51] Youcheng Sun, Xiaowei Huang, and Daniel Kroening. 2018. Testing Deep Neural Networks. *ArXiv* abs/1803.04792 (2018).

[52] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, M. Kwiatkowska, and Daniel Kroening. 2018. Concolic Testing for Deep Neural Networks. *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)* (2018), 109–119.

[53] Anshuman Suri and David Evans. 2020. One Neuron to Fool Them All. *arXiv preprint arXiv:2003.09372* (2020).

[54] Yuchi Tian, Kexin Pei, Suman Sekhar Jana, and Baishakhi Ray. 2018. DeepTest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars. *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)* (2018), 303–314.

[55] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. 2019. Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks. *2019 IEEE Symposium on Security and Privacy (SP)* (2019), 707–723.

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

[57] Jialai Wang, Han Qiu, Yi Rong, Hengkai Ye, Qi Li, Zongpeng Li, and Chao Zhang. 2022. Bet: black-box efficient testing for convolutional neural networks. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. 164–175.

[58] Zan Wang, Hanmo You, Junjie Chen, Yingyi Zhang, Xuyuan Dong, and Wenbin Zhang. 2021. Prioritizing Test Inputs for Deep Neural Networks via Mutation Analysis. *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)* (2021), 397–409.

[59] Michael Weiss and Paolo Tonella. 2022. Simple techniques work surprisingly well for neural network test prioritization and active learning (replicability study). *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis* (2022). https://api.semanticscholar.org/CorpusID:248496352

[60] Samuel Wilson, Tobias Fischer, Feras Dayoub, Dimity Miller, and Niko Sünderhauf. 2023. SAFE: Sensitivity-aware features for out-of-distribution object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 23565–23576.

[61] Dongxian Wu and Yisen Wang. 2021. Adversarial Neuron Pruning Purifies Backdoored Deep Models. In *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (Eds.). https://openreview.net/forum?id=4cEapqXfP30

[62] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. arXiv:cs.LG/1708.07747 [cs.LG]

[63] Xiaofei Xie, Tianlin Li, Jian Wang, L. 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 (TOSEM)* 31 (2022), 1 – 27.

[64] Xiaofei Xie, L. Ma, Felix Juefei-Xu, Hongxu Chen, Minhui Xue, Bo Li, Yang Liu, Jianjun Zhao, Jianxiong Yin, and S. See. 2018. DeepHunter: Hunting Deep Neural Network Defects via Coverage-Guided Fuzzing. *arXiv: Software Engineering* (2018).

[65] Shenao Yan, Guanhong Tao, Xuwei Liu, Juan Zhai, Shiqing Ma, Lei Xu, and X. Zhang. 2020. Correlations between deep neural network model coverage criteria and model quality. *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (2020).

[66] Chongzhi Zhang, Aishan Liu, Xianglong Liu, Yitao Xu, Hang Yu, Yuqing Ma, and Tianlin Li. 2019. Interpreting and Improving Adversarial Robustness of Deep Neural Networks With Neuron Sensitivity. *IEEE Transactions on Image Processing* 30 (2019), 1291–1304. https://api.semanticscholar.org/CorpusID:212900752

[67] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P. Xing, Laurent El Ghaoui, and Michael I. Jordan. 2019. Theoretically Principled Trade-off between Robustness and Accuracy. *ArXiv* abs/1901.08573 (2019).

[68] Jianping Zhang, Weibin Wu, Jen tse Huang, Yizhan Huang, Wenxuan Wang, Yuxin Su, and Michael R. Lyu. 2022. Improving Adversarial Transferability via Neuron Attribution-based Attacks. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022), 14973–14982. https://api.semanticscholar.org/CorpusID:247922429

[69] Lingming Zhang. 2018. Hybrid Regression Test Selection. *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)* (2018), 199–209.

[70] Tahereh Zohdinasab, Vincenzo Riccio, Alessio Gambi, and Paolo Tonella. 2023. Efficient and Effective Feature Space Exploration for Testing Deep Learning Systems. *ACM Transactions on Software Engineering and Methodology* 32 (2023), 1 – 38. https://api.semanticscholar.org/CorpusID:250118043