Neuron Semantic-Guided Test Generation for Deep Neural Networks Fuzzing

LI HUANG* and WEIFENG SUN*, Chongqing University, China
MENG YAN†, Chongqing University, China
ZHONGXIN LIU, Zhejiang University, China
YAN LEI, Chongqing University, China
DAVID LO, Singapore Management University, Singapore

In recent years, significant progress has been made in testing methods for deep neural networks (DNNs) to ensure their correctness and robustness. Coverage-guided criteria, such as neuron-wise, layer-wise, and path-/trace-wise, have been proposed for DNN fuzzing. However, existing coverage-based criteria encounter performance bottlenecks for several reasons: ❶ **Testing Adequacy**: Partial neural coverage criteria have been observed to achieve full coverage using only a small number of test inputs. In this case, increasing the number of test inputs does not consistently improve the quality of models. ❷ **Interpretability**: The current coverage criteria lack interpretability. Consequently, testers are unable to identify and understand which incorrect attributes or patterns of the model are triggered by the test inputs. This lack of interpretability hampers the subsequent debugging and fixing process. Therefore, there is an urgent need for a novel fuzzing criterion that offers improved testing adequacy, better interpretability, and more effective failure detection capabilities for DNNs.

To alleviate these limitations, we propose NSGen, an approach for DNN fuzzing that utilizes neuron semantics as guidance during test generation. NSGen identifies critical neurons, translates their high-level semantic features into natural language descriptions, and then assembles them into human-readable DNN decision paths (representing the internal decision of the DNN). With these decision paths, we can generate more fault-revealing test inputs by quantifying the similarity between original test inputs and mutated test inputs for fuzzing. We evaluate NSGen on popular DNN models (VGG16_BN, ResNet50, and MobileNet_v2) using CIFAR10, CIFAR100, Oxford 102 Flower, and ImageNet datasets. Compared to 12 existing coverage-guided fuzzing criteria, NSGen outperforms all baselines, increasing the number of triggered faults by 21.4% to 61.2% compared to the state-of-the-art coverage-guided fuzzing criterion. This demonstrates NSGen's effectiveness in generating fault-revealing test inputs through guided input mutation, highlighting its potential to enhance DNN testing and interpretability.

CCS Concepts: • **Software and its engineering** → *Software testing and debugging*.

Additional Key Words and Phrases: Deep learning testing, test input generation, fuzzing

1 INTRODUCTION

Deep neural networks (DNNs) have witnessed remarkable advancements over the past few decades and are now extensively employed in diverse applications such as image classification [19], computer vision [31],

*Both authors contributed equally to this research.
†corresponding author

Authors' addresses: Li Huang, lee.h@cqu.edu.cn; Weifeng Sun, weifeng.sun@cqu.edu.cn, Chongqing University, China; Meng Yan, Chongqing University, ChongQing, China, mengy@cqu.edu.cn; Zhongxin Liu, Zhejiang University, China, liu_zx@zju.edu.cn; Yan Lei, Chongqing University, China, yanlei@cqu.edu.cn; David Lo, Singapore Management University, Singapore, davidlo@smu.edu.sg.

ACM 1557-7392/2024/8-ART
https://doi.org/10.1145/3688835

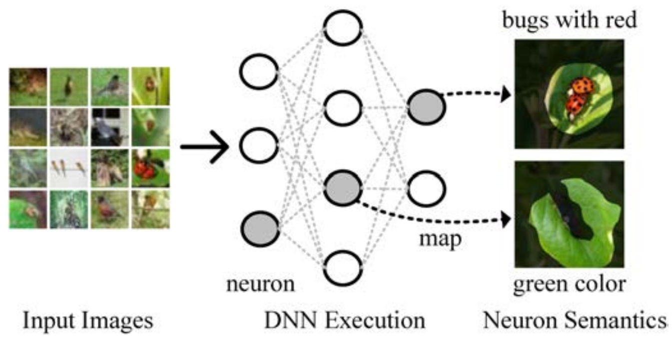


Fig. 1. Examples of the neuron-descriptions pairs.

speech recognition [2], natural language processing [20], and medical diagnosis [18, 40]. Despite their impressive performance, concerns about the safety and robustness of DNNs have been raised, especially in safety-critical applications like autonomous driving [3]. Any unexpected misbehavior of DNNs may lead to catastrophic consequences, making it essential to test DNNs and effectively identify their defects.

Fuzzing, as a well-established automatic testing technique [9], has been demonstrated to be effective in detecting bugs and vulnerabilities in traditional software systems. Fuzzing involves generating random test inputs for the software under test, monitoring the application's behavior during test execution, collecting test inputs' execution information, and mutating the inputs to trigger faults based on specific coverage criteria. While traditional coverage-guided fuzzing (CGF) methods [12, 17, 65] are effective, applying such techniques directly to DNNs poses challenges due to inherent differences in test input mutation strategies, and feedback guidance between DNNs and traditional software. Therefore, the design of an effective fuzzing strategy becomes crucial in the context of DNN testing, with feedback guidance, i.e., coverage criteria, playing a pivotal role in effectively uncovering faults in DNNs.

To this end, several neural coverage criteria have been proposed for DNNs, based on the neural activation status [59]. Such criteria can be broadly categorized into neuron-wise, layer-wise, and trace-/path-wise criteria. Neuron-wise criteria [45, 59] evaluate test input coverage by considering each neuron individually, while layer-wise criteria [45] assess coverage from a layer-level perspective. Trace-/path-wise criteria [37, 39, 81] measure DNN coverage based on the traces or paths traversed by neurons. These criteria encompass various coverage calculations, such as neuron coverage (NC) [59], top-k neuron coverage (TKNC) [45], and neuron path coverage (NPC) [81]. Guided by existing coverage criteria, automated testing techniques like DeepXplore [59], DeepTest [77], and DeepHunter [82] have been developed to generate test inputs that maximize above-mentioned neural coverage. However, recent studies [30, 47, 69, 87, 88] have highlighted the limitations of existing neural coverage criteria in guiding the generation of DNN test inputs: ① ***Testing adequacy***. It has been observed that a partial neural coverage criterion could achieve full coverage using only a small number of test inputs. In this case, increasing the number of test inputs did not consistently improve the quality of models [30, 47]. ② ***Interpretability***. The existing coverage criteria lack interpretability [47, 69]. As a result, testers are unable to identify what incorrect attributes or patterns of the model are triggered by the test inputs. Additionally, this lack of interpretability hinders the subsequent debugging and fixing process, as it becomes challenging to understand and address the underlying bugs in the model.

To alleviate the above problems, we present a novel DNN test input generation method called NSGen (Neuron Semantic-Guided Test Generation) for DNNs fuzzing. NSGen starts by extracting the semantics of all neurons from the DNN under test and translating them into natural language descriptions. Next, for a given original image and its mutated counterpart, we identify neurons that significantly influence the DNN's prediction results. We construct a decision path, which represents the internal decision of the DNN on given inputs [81], comprising neuron semantics by assembling the corresponding natural language descriptions according to a template. Finally, we measure the similarity between the original and mutated images based on their decision path described by natural language. Intuitively, a mutated image showing significant dissimilarity in the decision path is more effective in uncovering violations and exposing flaws. Such images explore different DNN decision paths compared to the original input. Figure 1 shows that the intricate semantic information, acquired by the neurons within a DNN, can be mapped into expressive and comprehensible natural language descriptions. By associating natural language descriptions with individual neurons, NSGen enables a deeper understanding of the semantic information they represent. These descriptions reveal specific semantic or structural information that neurons activate in response to input data, facilitating the explanation of DNN's inner workings. Additionally, this neuron-level semantic information enhances our comprehension of how the model makes decisions based on various features extracted from the input data [32, 53, 92].

We evaluate the effectiveness of NSGen by extensively testing it on three models used in the latest DNN fuzzing paper [90], including VGG16_BN, ResNet50, and MobileNet_v2, with four datasets CIFAR10, CIFAR100, Oxford 102 Flower, and ImageNet, for the task of image classification. Compared to 12 existing coverage-guided fuzzing criteria, NSGen outperforms all baselines, significantly increasing the number of triggered faults by 21.4% to 61.2% compared to the state-of-the-art coverage-guided fuzzing criterion. In summary, our contributions can be summarized as follows:

- **Approach**. We introduce NSGen, an effective neuron semantic-guided test generation approach specifically designed for DNN fuzzing. This pioneering study represents one of the earliest endeavors in the domain of neuron semantic guided test generation for DNNs fuzzing.
- **Interpretation**. Our approach contributes to the comprehension and interpretability[1] of DNN models by mapping semantic information of influential neurons into natural language descriptions and forming decision paths based on neuron semantics.
- **Study**. We demonstrate the effectiveness of NSGen on three typical models across four widely used datasets. NSGen yields a significant improvement in the diversity and fault revelation of the generated test inputs.

2 BACKGROUND

2.1 Deep Neural Networks

This paper focuses on DNNs for single-label classification. The formalization of a DNN classifier can be expressed as a function $f : X \rightarrow Y$, wherein it maps a set of input values X to a set of corresponding labels denoted as Y. The output generated by the DNN classifier assumes the form of a probability distribution $P(Y|X)$, which conveys the likelihood that an input vector $x \in X$ belongs to each class represented in Y. Subsequently, the label assigned to the input x corresponds to the class with the highest probability.

A typical DNN classifier, denoted as f, comprises an input layer, a series of hidden layers, and an output layer, each composed of multiple neurons. The parameters θ associated with the DNN represent the weights assigned to the edges connecting neurons between adjacent layers. When provided with an input vector x, the DNN's output, denoted as $f_\theta(x)$, can be calculated as the weighted sum of outputs from all the neurons.

The training process for a DNN classifier involves a training dataset $D = \{(x_i, y_i)\}_{i=1}^N$, which includes a collection of N input examples, denoted as $\{x_1, x_2, \ldots, x_N\}$, along with their corresponding ground-truth labels, $\{y_1, y_2, \ldots, y_N\}$. The DNN classifier aims to optimize the following objective function:

[1]We say that NSGen is interpretable, as it is designed to identify the key neurons that contribute more ion the decision of the model based on the interpretation technique, i.e., Integrated Gradients [24, 75].

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i) \tag{1}$$

Here, $\mathcal{L}$ represents the loss function, used to assess the penalties for incorrect classifications. The DNN undergoes a training process by minimizing the loss on the training dataset and iteratively adjusting the parameters θ accordingly. Given the intricate non-linear computations inherent in DNNs, understanding the rationale behind their decision-making processes often poses a significant challenge.

2.2 Coverage-guided Testing

Coverage-guided testing (CGT) stands out as a widely embraced technique for the identification of software bugs [5]. Among the CGT methodologies, fuzzing, exemplified by AFL [91], stands as a prominent approach that has successfully unearthed thousands of bugs within real-world software [16, 70, 76]. The fuzzing process initiates by establishing a seed corpus, which comprises an initial set of seed inputs. Subsequently, in each iteration, a seed is selected from this corpus, and mutants are generated based on the chosen seed. The execution of mutants is accompanied by the collection of code coverage information, such as branch coverage. Mutants that enhance coverage, i.e., those that reveal new software behaviors, are incorporated into the seed queue. CGT is also tailored to test DNN with specific coverage criteria devised for DNNs [37, 45, 58, 90]. Numerous CGT techniques have been introduced for DNN testing, incorporating diverse coverage feedback mechanisms [29, 44, 56, 59, 77, 82]. Nevertheless, the explicit utilization of neural semantic information to steer fuzz testing remains an unresolved challenge.

2.3 Natural Language Description for Neurons

In this section, we present related concepts that are essential to the core of our approach, which involves mapping the semantic features of neurons into natural language descriptions. The Show, Attend, and Tell (SAT) model [84] is a framework used for generating natural language descriptions of images. The SAT model employs an image classifier g pre-trained on a large dataset, to extract k **a**nnotation vectors from a single input image x, representing different parts of the image:

$$\boldsymbol{A} = [\boldsymbol{a}_1; \boldsymbol{a}_2; \ldots; \boldsymbol{a}_k] \tag{2}$$

The annotation matrix $\boldsymbol{A}$ is input to a decoder LSTM, with the hidden state initialized based on the mean of the annotation matrix $\bar{\boldsymbol{a}} = 1/L \sum_i \boldsymbol{a}_i$. The decoder employs an additive attention mechanism [7] to focus on the features, and it utilizes the attenuated annotation matrix [33] along with the previous token to predict the subsequent token at each time step.

Expanding upon SAT, Hernandez et al. [32] presented a method for generating natural language descriptions of deep visual features, specifically focusing on convolutional layer neurons in DNNs. The method consists of two distinct steps. First, we need to determine the exemplar representation of each neuron, wherein each neuron is represented through the set of input regions on which its activation values surpass a fixed threshold γ (i.e., set it to the 0.99 percentile of activations for the neuron f_i). We define exemplar representation as follows:

Definition 1. *Consider a neural network $f : X \to Y$, where X represents the input space and Y represents the output space. Let $f_i(x)$ denote the activation value of the i-th neuron in network f given an input x from the input space. In this context, we define a neuron representation (referred to as $\boldsymbol{R}$) as follows:*

$$\boldsymbol{R}_i = \{x \in X : f_i(x) > \gamma_i\} \tag{3}$$

With this explicit representation of neuron f_i, the next step is to create a description $\boldsymbol{d}_i$, where $\boldsymbol{d}_i$ denotes the encoded vector of a neuron's description. In computer vision applications, $\boldsymbol{R}_i$ is typically a set of images. Although

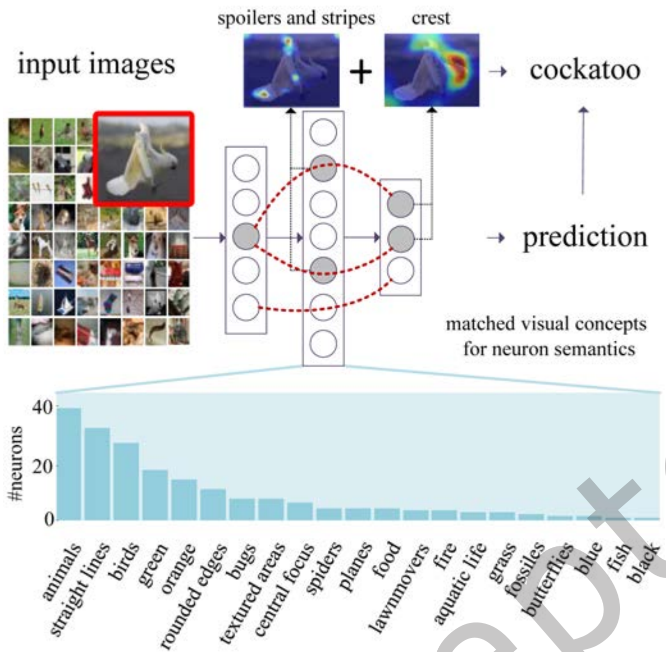


Fig. 2. An example of the neuron semantics.

humans can easily describe such images, directly optimizing $\boldsymbol{d}_i := argmax_i\, p(\boldsymbol{d}|\boldsymbol{R}_i)$[2] may not be the most effective approach due to the semantic gap, and complexity in capturing the specific behavior of computer vision models. The generated neuron description should capture the specificity of neuron function, particularly in relation to other neurons in the same model. Therefore, Hernandez et al. [32] optimize the pointwise mutual information [14] between descriptions $\boldsymbol{d}_i$ and representation sets $\boldsymbol{R}_i$.

Definition 2. *The description of the neuron f_i in terms of maximum mutual information is defined as follows:*

$$d(f_i) = \arg\max_{\boldsymbol{d}_i} \left(\log p(\boldsymbol{d}_i|\boldsymbol{R}_i) - \log p(\boldsymbol{d}_i)\right) \tag{4}$$

In sum, to search fine-grained natural language descriptions for neurons, we maximize the pointwise mutual information of the image regions where the neurons are active.

3 MOTIVATION

Currently, fuzzing techniques heavily rely on neuron coverage as a guiding metric to assess the diversity and adequacy of the test set. The prevailing coverage criteria prioritize maximizing coverage to allow fuzzing-generated input samples to explore the model's internal state comprehensively, effectively identifying potential bugs and vulnerabilities.

However, existing neuron coverage criteria lack interpretability, making it unclear which semantic features are captured by activated neurons. This lack of interpretability hinders the understanding of how the model processes and leverages semantic information from different neurons.

[2] $p(\boldsymbol{d}|\boldsymbol{R}_i)$ is the possibility of using natural language descriptions $\boldsymbol{d}$ to describe the images $\boldsymbol{R}_i$. Given the most active images corresponding to a neuron, the corresponding natural language description is obtained by manually summarizing all the content displayed within the highlighted area in the images.

Furthermore, Yuan et al. [90] demonstrate that neurons in DNN collaborate to comprehend the semantics of high-dimensional inputs (e.g., images). This collaboration may lead to existing coverage covering some activation states/paths that have similar or even identical neuron semantics despite their large activation differences from those that have been previously covered, leading to redundant generated test samples. This limitation becomes apparent when the number of test samples is restricted, thus hindering the ability to thoroughly test the DNN and challenge its decision boundaries. Figure 2 provides an illustrative example of a Class Activation Map (CAM) [96] learned by neurons from different layers, describing various semantic information aspects related to the object "cockatoo." The CAM effectively captures the bird's salient features, including spoilers, stripes, and crest. Notably, during model training, specific neurons interact to learn distinct features or concepts [10, 32]. Consequently, for a certain test input (i.e., cockatoo), different neurons may be activated within the same layer, but their CAMs could overlap, indicating consistent neuron semantics. To bolster this finding, the statistical plot (shown at the bottom) in Figure 2 displays the semantic statistics of neurons obtained by decomposing one convolutional layer with netdissect[3]. The plot demonstrates that multiple neurons co-encode similar or identical semantic concepts, corroborating previous studies [28, 73]. These findings suggest that existing neuron coverage criteria may result in the formation of decision paths with similar or identical neuron semantics. The current coverage criteria predominantly emphasize the number of activated neurons [90] (or cover more decision paths like the red dotted lines in Figure 2), but overlook the specific semantic features recognized by individual neurons. In other words, it lacks the capability to discern and account for the consistency of semantic information represented by different activated neurons, potentially leading to redundancy in the generated test samples.

In light of the aforementioned issues, namely:

(1) ***Lacking the interpretability,***
(2) ***Existing redundant semantic information of neurons,***

To address the above issue, we introduce a guidance criterion for DNN fuzzing named Decision Path Discrepancy (DPD).

Definition 3. *Consider a neural network f, an original image set O and a corresponding mutated image set M, for each pair of original image o_i and mutated image m_i, we generate their decision paths $P(o_i)$, $P(m_i)$, then we get DPD as follows:*

$$DPD(O, M) = \frac{1}{|O|} \sum_{i=1}^{|O|} \mathbb{I}[D(P(o_i), P(m_i)) < \tau] \tag{5}$$

Here, $\mathbb{I}$ represents an indicator function, D is a measure based on semantic similarity techniques, and τ is a predetermined threshold used to determine if the semantic discrepancy between two decision paths is significant. If $D(P(o_i), P(m_i))$ is less than τ, it indicates a significant semantic discrepancy between the two decision paths.

Unlike traditional neuron coverage criteria that merely quantify the activation of neurons, DPD could identify critical neurons and employ natural language descriptions to elucidate the semantic information of visual concepts they capture. This method not only provides a deeper insight into the decision-making process of DNNs but also ensures a more meaningful assessment of test sets by focusing on the semantic discrepancy between decision paths. Consequently, DPD encourages the generation of more diverse and interpretable test inputs that effectively probe the model's decision boundaries and uncover potential faults with a nuanced understanding of model behavior.

4 METHODOLOGY

In this section, we elaborate on the proposed neuron semantic guided test generation approach for DNN fuzzing. We take an overview of NSGen and then describe each of its key components in detail.

[3]Netdissect [9] matches the pattern of neuron activations to the pattern of a pre-defined label mask

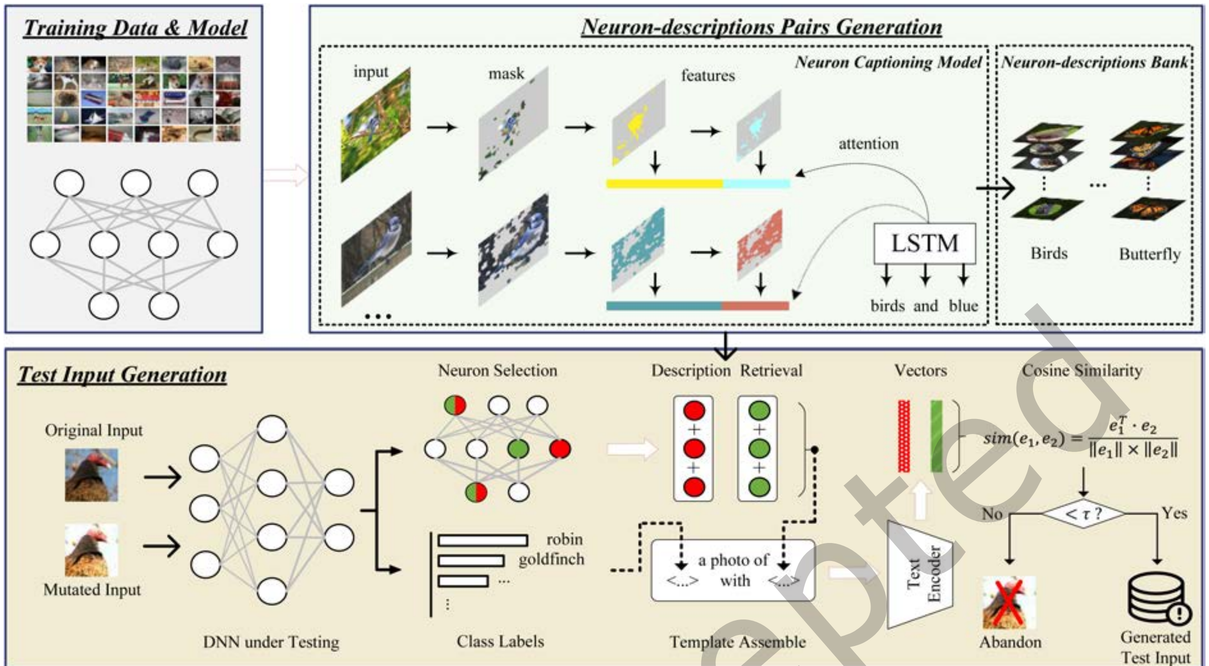


Fig. 3. The architecture of NSGen.

4.1 Overview of NSGen

Figure 3 illustrates the overall architecture of NSGen, and Algorithm 1 specifies the details. NSGen drives fuzzing via Decision Path Discrepancy (DPD), which consists of two key components: neuron-description pairs generation and test input generation. We begin by generating neuron-description pairs, as illustrated in Figure 1. Next, we select neurons for both the original image and the mutated image based on their contributions to the prediction results of the tested DNN. Finally, we assemble the neuron descriptions and map them into the text-visual multimodal space to calculate the similarity.

The detailed workflow of NSGen is described in Algorithm 1. We utilize the similarity metric *sim* as the guiding criterion to direct the input mutation process. The inputs to NSGen include the mutation rules $\mathcal{T}$ (*c.f.* Section 5.2), fuzzing seed $\mathcal{S}$, the number of trials, Contrastive Language-Image Pre-Training model (CLIP[4]) as text encoder, and the tested DNN $\mathcal{D}$. Before initiating the fuzzing loop, we first set the similarity threshold (Line 2, *c.f.* Section 4.3.4) and generate neuron-description pairs (Line 3, *c.f.* Section 4.2) for subsequent use. During the fuzzing process, given the limited testing resources, we set a termination condition, e.g., when the number of iterations of the algorithm reaches 10K or the running time exceeds 6 hours (Line 4). Once the termination condition is reached, the fuzzing process stops. Otherwise, the fuzzer samples from the fuzzing seed (Line 5) and mutates (Lines 6-8). Since random mutations may yield meaningless seeds, the fuzzer judiciously assesses the validity of each mutated seed (Line 10). For each valid mutant derived from the original seed, NSGen retrieves the corresponding neuron description from the neuron-description pairs bank (Lines 11-14, *c.f.* Section 4.3.1). Subsequently, a template is constructed, and the cosine similarity between the mutant and the original seed is calculated. The similarity score is then validated against a predetermined threshold to determine whether the mutant qualifies as an expected test input (Lines 15-21, *c.f.* Section 4.3.2 and Section 4.3.3).

[4]CLIP [62] is a neural network trained on a variety of (image, text) pairs. https://github.com/openai/CLIP.

Algorithm 1 : Neuron semantic-guided test input generation

Require: mutation rules $\mathcal{T}$, fuzzing seed $\mathcal{S}$, *num*, tested DNN $\mathcal{D}$, pre-trained language-image model $\mathcal{CLIP}$
Ensure: a set of generated inputs F
$F \leftarrow \emptyset$
// set similarity threshold (*c.f.* Section.4.3.4)
$\tau = setThreshold()$
Generate neuron-description pairs $\mathcal{B}$ by Equation (4)
while not terminate() **do**
  $s = sample(\mathcal{S})$
  for each $i \in 1 .. num$ **do**
    $t = sample(\mathcal{T})$
    $\hat{s} \leftarrow t(s)$
    // assess the validity of mutated seeds (*c.f.* Section.5.2)
    if $is\_valid(\hat{s}, s)$ **then**
      // select neurons and return the index of neurons
      $s\_id, \hat{s}\_id = selectNeuron(\mathcal{D}, s, \hat{s})$
      // retrieve descriptions from $\mathcal{B}$ by index
      $d_s, d_{\hat{s}} = retrieveDescription(\mathcal{B}, s\_id, \hat{s}\_id)$
      // construct templates (*c.f.* Section.4.3.2)
      $e_s, e_{\hat{s}} = \mathcal{CLIP}(d_s, d_{\hat{s}})$
      $sim = Cosine(e_s, e_{\hat{s}})$
      if $sim < \tau$ **then**
        $\mathcal{S}.add(\hat{s})$
        $F \leftarrow F \cup \{\hat{s}\}$
        Break
      end if
    end if
  end for
end while

4.2 Neuron-Description Pairs Generation

In this section, we will generate and store the descriptions of each neuron in the DNN. In this section, we will generate and store the descriptions of each neuron in the DNN. The process is concretized with Figure 4.

4.2.1 Build Exemplar Representations of Neurons. As described in Section 2.3, for each neuron, we select the k top-activating images (i.e., neuron representation $\boldsymbol{R}$), where $k = 15$, following the same setting as the previous paper [32] for describing the neuron semantics. Additionally, for each activated image, we assign a corresponding mask m_j to emphasize the regions of highest activation. As illustrated in Figure 4, a series of bird images (top-activating images) and their corresponding masks are presented. Each mask accurately highlights the areas in the image with the most significant neuronal activity, such as the beak and legs. Our goal is to characterize the semantics of neurons based on the highlighted regions of the k top-activating images. The procedure involves the following detailed steps:

1) Decide sets of images: In the traditional SAT model [84], the k features represent distinct spatial localities within a single image. In contrast, in our approach, each feature $\boldsymbol{a}_j$ is associated with a top-activating input image x_j.

2) Highlight regions of greatest activation: For each of the top-activating images x_j, a corresponding mask m_j is available, which highlights the regions of greatest activation in the image. To integrate these masks into the pooling function, we downsample each mask m_j using bilinear interpolation [72] to match the spatial shape of the corresponding feature map $g_l(x_j)$. The downsampled mask is denoted as $downsample(m_j)$. Next, we apply the mask to each channel c at layer l by element-wise multiplication (symbolized as $\odot$) with $downsample(m_j)$, resulting in a length c vector. Finally, we perform spatial summation along each channel to obtain the pooled features. This process can be described as follows, vec denotes vector form:

$$pooling_c(m_j, g_l(x_j)) = \mathbf{1}^\top \mathrm{vec}(downsample(m_j) \odot g_{l,c}(x_j)) \tag{6}$$

3) Encode multiple resolutions: The annotation vector for the j-th image x_j is derived by pooling the features extracted from each convolutional layer of the pre-trained image encoder. More precisely, $g_l(x)$ represents the output of layer l in the encoder with a total of L layers, and $pool$ denotes a pooling function that leverages the mask to aggregate the features. The annotation vector for the j-th image x_j is as follows:

$$\boldsymbol{a}_j = \left[pooling(m_j, g_1(x_j)); \ldots ; pooling(m_i, g_L(x_j))\right] \tag{7}$$

Each $\boldsymbol{a}_j$ is thus a length $\sum_l C_l$ vector, where C_l represents the number of channels at layer l of g. Now, we obtain annotation matrix $\boldsymbol{A}$, namely:

$$\boldsymbol{A} = [\boldsymbol{a}_1; \ldots ; \boldsymbol{a}_j; \ldots ; \boldsymbol{a}_k] \tag{8}$$

As depicted in Figure 4, annotation matrix $\boldsymbol{A}$ will be used as input to the decoder attention mechanism.

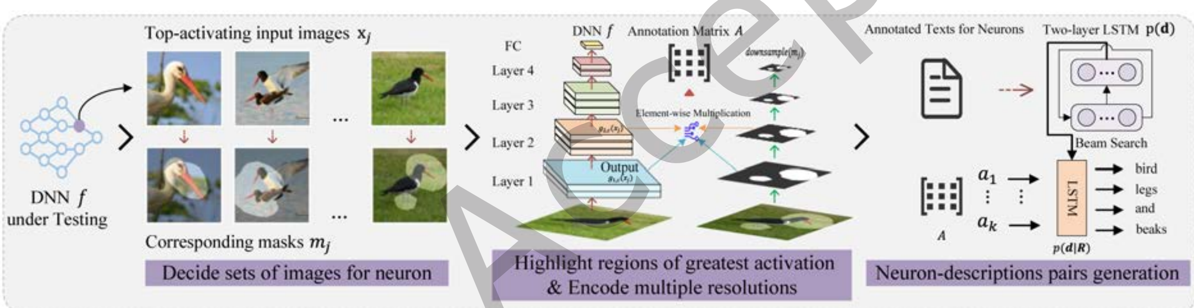


Fig. 4. The workflow of Neuron-Descriptions Pairs Generation.

4.2.2 Neuron-Descriptions Pairs Generation. In this step, we decode and rank the natural language descriptions of the neurons according to Equation (4) as depicted in Figure 4, involving two decoding processes. We compute the probability $p(\boldsymbol{d}|\boldsymbol{R})$ that humans use natural language descriptions d to describe the image region (i.e., the k top-activating images), and the probability $p(\boldsymbol{d})$ of using descriptions $\boldsymbol{d}$ to describe any neuron.

To determine the human usage of natural language description $\boldsymbol{d}$ for image regions, we construct the required annotation matrix $\boldsymbol{A}$ for the SAT model [84] as discussed in Section 4.2.1. Subsequently, we employ the same process as the SAT model to decode $p(\boldsymbol{d}|\boldsymbol{R})$. This involves using a single LSTM with an input embedding size of 128 and a hidden size of 512. Additionally, for the attention mechanism within the SAT model, we linearly map the current hidden state and the annotation matrix to vectors of size 512 before computing the attention weights. For human usage in describing any neuron with description $\boldsymbol{d}$, we use a two-layer LSTM [33] to represent $p(\boldsymbol{d})$. This involves an input embedding size of 128, a hidden state size of 512, and a cell size of 512. We use an open-source, manually annotated dataset MILANNOTATIONS [32] to train these two decoders. This dataset comprises representation sets $\boldsymbol{R}$ derived from neurons across seven vision models (including classification models like ResNet152 [31] and generative models like BigGAN [15]) trained on two large datasets (ImageNet and

Places365 [97]), encompassing a total of 20,000 neurons, each associated with a manually annotated natural language description $\boldsymbol{d}$. We use these $(\boldsymbol{R}, \boldsymbol{d})$ pairs to train the decoder $p(\boldsymbol{d}|\boldsymbol{R})$, and rely solely on human-annotated natural descriptions $\boldsymbol{d}$ to train the decoder $p(\boldsymbol{d})$. And specific training details can be found at [1]. This extensive dataset ensures the generalizability of our approach across various model architectures and complex datasets, while our experiments (*c.f.* Section 7.7) further support this conclusion.

During the process of decoding descriptions, the search is constrained to a set of captions with high probabilities under $p(\boldsymbol{d}|\boldsymbol{R})$, and these are ranked according to Equation (4) [46]. Specifically, we conduct a beam search on $p(\boldsymbol{d}|\boldsymbol{R})$ and use the full beam (i.e., set beam size to 50) after the final search step as a set of candidate descriptions. From this set, we select the top-1 description.

4.3 Test Input Generation

4.3.1 Neuron Selection. After generating and storing the descriptions, in this section, we focus on selecting the neurons that hold significance for the input. A neuron is deemed important if it significantly influences the model's prediction results on the input, i.e., its output demonstrates a high contribution value. To identify such critical neurons, we leverage integrated gradients [24, 75] to compute each neuron's contribution score to the final prediction.

In line with Definition 1, let us consider a DNN f and an input image $x \in \mathbb{R}^n$, along with an empty input image $x' \in \mathbb{R}^n$. To compute integrated gradients, we calculate gradients at various points along a straight-line path from x' to x and then integrate these gradients. By accumulating these micro explanations, integrated gradients capture the net difference between the prediction score at the baseline and that at the input x, elucidating how the function f varies from the informationless baseline to its final value. Formally, the integrated gradient for the ϵ^{th} base feature (e.g., a pixel) of an input x and baseline x' can be defined as:

$$\nabla_\epsilon(x) = (x_\epsilon - x'_\epsilon) \cdot \int_{\omega=0}^{1} \frac{\partial f(x' + \omega(x - x'))}{\partial x_\epsilon} d\omega \tag{9}$$

Here, $\frac{\partial f(x)}{\partial x_\epsilon}$ represents the gradient of f along the ϵ^{th} dimension at x.

Integrated gradients only provide attributions for base features, such as the pixels in an object recognition network. Note that our goal is to calculate the contribution of hidden layer neurons to the final prediction result. Therefore, let's consider a specific neuron f_i in a hidden layer of a DNN, the contribution $\phi_\epsilon^{f_i}$ for an input point ϵ (i.e., a pixel) can be computed as follows:

$$\phi_\epsilon^{f_i}(x) = (x_\epsilon - x'_\epsilon) \cdot \int_{\omega=0}^{1} \frac{\partial f(x' + \omega(x - x'))}{\partial f_i} \cdot \frac{\partial f_i}{\partial x_\epsilon} d\omega \tag{10}$$

Now, we can define the mean contribution for the hidden neuron (e.g., a single filter) as:

$$\phi^{f_i}(x) = \frac{1}{Height_{f_i} \times Width_{f_i}} \sum_{Height_{f_i}} \sum_{Width_{f_i}} \sum_{\epsilon} \phi_\epsilon^{f_i}(x) \tag{11}$$

With the calculation method described above, we can determine the contribution of neurons in convolutional layers for the original (mutated) images. However, to reduce the computational cost, we only obtain the attributions for neurons in the selected k layers of the DNN, aligning with the setting proposed in [10]. In particular, the k values are determined as follows: For ResNet50, VGG16_BN, and MobileNet_v2, the values of k are set to 5, 5, and 10, respectively. In Section 6.4.3, we discuss the impact of different k values on NSGen. Note that for each selected convolutional layer, we only choose neurons with a top-1 contribution score.

4.3.2 Template Construction. Once we have identified the important neurons for each input, the next step involves constructing a template to assemble the corresponding descriptions of these neurons. Intuitively, incorporating descriptions of visual contexts can provide valuable additional information about the test input, leading to more precise natural language descriptions. For example, adding “in the grass” to “a photo of a cat” can help distinguish it from other photos of cats taken in different contexts [64]. Therefore, we assemble the natural language descriptions of the test input from two perspectives: the class labels predicted by the model and the descriptions of some critical neurons. The neuron descriptions offer insights into the semantic features on which the model’s predictions are based, while the class labels provide contextual information. To construct text prompts, we utilize the template “a photo of $< \ldots >$ with $< \ldots >$.” The former placeholder represents the class labels, while the latter encompasses the natural language descriptions of neurons. Specifically, we shuffle the order of the class labels predicted by the model and connect them using the conjunction “or”, and for retrieved neuron descriptions from the description bank, we concatenate them in the order of DNN layers with commas. Thus far, we have constructed the neuron semantics of the test input into a neuron semantic decision path.

4.3.3 Similarity Calculation. After assembling the neuron semantic decision paths for both the original image $\boldsymbol{t}_{org}$ and its mutated counterpart $\boldsymbol{t}_{mut}$ into templates, we utilize CLIP as the default text encoder (refer to Section 6.3 for a detailed comparison with other text encoders) to map these templates into the text-visual multimodal space. This allows us to calculate fine-grained similarity between the decision paths. We calculate the cosine similarity using the following formulas:

$$sim(\boldsymbol{e}_{org}, \boldsymbol{e}_{mut}) = \frac{\boldsymbol{e}_{org}^T \cdot \boldsymbol{e}_{mut}}{\|\boldsymbol{e}_{org}\| \times \|\boldsymbol{e}_{mut}\|} \tag{12}$$

Here, the vectors $\boldsymbol{e}_{org}$ and $\boldsymbol{e}_{mut}$ represent the template of $\boldsymbol{t}_{org}$ and $\boldsymbol{t}_{mut}$ after the template construction process, respectively.

4.3.4 Hyper-parameter Setting. For the selection of the hyperparameter τ, we follow a systematic approach. Specifically, we begin by randomly sampling 1% of each class from the CIFAR10/CIFAR100 training sets and 1‰ of each class from the ImageNet training set. For the Oxford 102 Flower dataset, which has a smaller overall dataset size, we sample a larger proportion, 50%, to ensure sufficient coverage. After sampling, each selected image undergoes mutation through our entire mutation space [5]. This is accomplished using 95 distinct mutation rules that we have predefined (refer to our code [1]). These mutation rules (*c.f.* Section 5.2) encompass a range of pixel-level adjustments, affine transformations, and style transfers, each applied with fixed parameters to ensure consistency and meaningful variance. This procedure allows us to extensively evaluate the model’s sensitivity to varied input perturbations, ensuring that the testing is comprehensive. Through a detailed analysis of the distribution of similarity scores obtained, we determine the threshold τ based on the lower quartile of the similarities. The threshold values are determined as follows: $\tau = 0.81$ for CIFAR10, $\tau = 0.71$ for CIFAR100, $\tau = 0.87$ for Oxford 102 Flower, and $\tau = 0.72$ for ImageNet. This meticulous approach to hyperparameter selection ensures the reliability and relevance of the threshold values.

5 IMPLEMENTATION

We have conducted extensive experiments to evaluate the performance of NSGen. This section introduces the experiment settings. To conduct the experiments, we implement NSGen upon Python 3.8.13 and PyTorch (ver. 1.11.0). All experiments are performed on a Ubuntu 20.04.1 LTS with Nvidia GeForce RTX 3090 (GPU), Intel

[5]For CIFAR10, CIFAR100, and Oxford 102 Flower datasets, the mutation space can be traversed in 2 hours, while for the ImageNet dataset, it takes less than 4 hours.

XEON 6226R 2.90GHz (CPU), 32GiB DDR4-3200 (Memory). To facilitate result verification and enable comparison with future research, we have made the code and data available at [1].

5.1 Datasets and DNN Models

We choose four widely recognized and publicly available datasets for evaluation in our study, namely, CIFAR10 [42], CIFAR100 [42], Oxford 102 Flower [55], and ImageNet [22] (as presented in Table 1). For each of these datasets, we have focused on well-established DNN models that have been extensively utilized in prior research [81, 82, 89, 90]. Our research places a significant focus on the generation of test inputs within the context of complex DNNs and datasets, with a keen examination of the scalability and practicality of NSGen.

CIFAR10 and **CIFAR100** both fall under the category of general-purpose image classification datasets. Each dataset includes a total of 60,000 images, with 50,000 used for training and 10,000 for testing. The images in both datasets are three-channel RGB images, each measuring 32 × 32 × 3 in dimensions. CIFAR10 consists of 10 distinct classes, while CIFAR100 presents a more complex challenge by featuring 100 different classes. To obtain competitive performance on CIFAR10 and CIFAR100, we study three well-known DNN models (i.e., VGG16_BN [71], ResNet50 [31], MobileNet_v2 [68]) as the subject models.

Table 1. Subject datasets and DNN models.

Dataset	Dataset Description	DNN Model	Top-1 Test Acc.
CIFAR10	General image with 10-class, image size is 32x32, the color is RGB	VGG16_BN*	93.81%
		ResNet50	93.78%
		MobileNet_v2	93.37%
CIFAR100	General image with 100-class, image size is 32x32, the color is RGB	VGG16_BN*	74.01%
		ResNet50	74.98%
		MobileNet_v2	74.29%
Oxford 102 Flower	Consisting of 102 flower categories, image size is 224x224, the color is RGB	VGG16_BN*	74.39%
		ResNet50	82.78%
		MobileNet_v2	83.02%
ImageNet	1000-class large-scale image class. image size is 128x128, the color is RGB	VGG16_BN*	73.36%
		ResNet50	75.69%
		MobileNet_v2	71.43%

*All image models have batch normalization (BN) layers.

Oxford 102 Flower is an image classification dataset consisting of 102 flower categories. It comprises a total of 8,189 images, with 1,020 allocated for training, 1,020 for validation, and 6,149 for testing. Each flower image is three-channel of size 224 × 224 × 3. This dataset has gained recognition for its remarkable diversity, as it exhibits a wide range of variations in terms of scale, pose, and lighting across its images. Additionally, it poses a unique challenge by encompassing categories that demonstrate notable intraclass variations, along with several categories that bear close relationships, thus constituting a fine-grained classification task [85]. Incorporating the Oxford 102 Flower dataset into our experiments serves a dual purpose. First, it allows us to evaluate NSGen's proficiency in effectively handling medium-sized datasets that feature a moderate number of classes. Second, it provides a platform for assessing NSGen's potential in generating neuronal semantic decision paths for similar samples.

ImageNet. We seek to provide further evidence of NSGen's scalability by venturing into the realm of practical-sized datasets. ImageNet is a noteworthy choice, as it serves as a benchmark in large-scale visual recognition challenges, specifically, the ILSVRC dataset [67], designed for general-purpose image classification. The intricacy of ImageNet lies in its vast training dataset, consisting of over one million instances, and a testing dataset comprising 50,000 samples. Additionally, the dataset features large data points, each with dimensions of 128 × 128 × 3 (roughly 16 times the dimensionality of CIFAR10/CIFAR100). Consequently, any automated testing tool faces a formidable challenge when dealing with ImageNet-sized datasets. In particular, our objective is to investigate whether NSGen could facilitate test input generation on the ImageNet, in conjunction with practical-sized DNN models[6], such as VGG16_BN and ResNet50.

5.2 Input Mutation Rules

In line with previous studies [89, 90], we use the mutation rules outlined in Table 2 to generate mutation images.

Pixel Level mutation scheme involves three operations to mutate input images and assess the robustness of DNNs [21, 56, 77]. These operations include changing the image brightness, adjusting the image contrast, and applying blurring through convolution with a sliding kernel.

Affine Type applies invertible transformations to mutate images, such as translating, rotating, scaling, and shearing. These transformations preserve the collinearity of objects, thereby retaining the correct semantics of images after mutation [77, 82].

Style Transfer is initially introduced by Zhang et al. [93] to transfer severe weather conditions from source images to target images, with a particular focus on driving scenes. Following the previous works [89, 90], we have integrated image style transfer as one of our mutation rules and extended its applicability to encompass general image style transfer. The style transfer mutation incorporates a wide variety of 5,935 styles, which allows us to create diverse input images while preserving the underlying semantic content.

Notable, to ensure the validity of the generated mutations and address the concerns about the repetition of original samples and potential image quality degradation, we have implemented specific controls in our mutation process: 1) The number of changed pixels in each mutated image is restricted to be less than α times the total number of pixels in the original image. This condition prevents excessive alterations that could lead to loss of semantic integrity. 2) The maximum change allowed in the value of any pixel is capped at β times 255, ensuring that changes are subtle and preserve the original image's visual coherence. In our experiments, we follow the setting taken in previous work [90] and set α to 0.2 and β to 0.4. These settings ensure that the mutations are not too extreme, preserving the essential characteristics of the original image while preventing quality degradation despite generating multiple mutations from a single sample, thereby introducing enough diversity to challenge the tested DNNs.

Table 2. Transformations $\mathcal{T}$ adopted.

Scheme	Original $s \rightarrow$ Mutated Image $\hat{s}$					Remark
Pixel Level	 s	$\rightarrow$	 blur	 brightness	 contrast	Robustness
Affine Type	 s	$\rightarrow$	 rotation	 scale	 translation	Shape bias
Style Transfer	 s	$\rightarrow$	 stylized1	 stylized2	 stylized3	Texture bias

[6]All ImageNet-trained models were officially provided by PyTorch. https://pytorch.org/vision/stable/models.html.

5.3 Baselines

In this section, we present a concise overview of the previous criteria utilized in coverage-guided fuzzing (CGF), which will be compared with NSGen. Additionally, we discuss the specific setups of each criterion.

• NC (Neuron Coverage) employs a threshold T that is applied to rescaled neuron outputs, restricting them to the range [0, 1] [58]. In our experiments, we set the threshold value T to 0.75 to determine whether a neuron is considered "covered" or "activated" for a particular test input.

• KMNC (K-Multisection Neuron Coverage) divides the range of normal neuron outputs into K sections. In our evaluation, we adopt $K = 100$, which is consistent with the settings used in previous studies [45, 81, 82].

• NBC/SNAC (Neuron Boundary Coverage/Strong Neuron Activation Coverage) does not require any specific parameters for its calculation. These criteria focus on different aspects of neuron activation and aim to measure the boundary regions of neuron activations and the strength of activations, respectively.

• TKNC/TKNP (Top-k Neuron Coverage/Top-k Neuron Patterns) consider a neuron as activated if it ranks among the top-K outputs among neurons within the same layer. In our evaluation, we set the values of K as 10 and 50, respectively, following the approach proposed by Ma et al [45]. These values were chosen to strike a balance between achieving good coverage and maintaining computational efficiency during the evaluation process.

• CC (Cluster-based Coverage) is a coverage criterion that involves a parameter T, which represents the distance threshold used to create clusters. As suggested in TensorFuzz [56] (Note that CC is the guideline criterion), we set the value of T to 10 for CIFAR10, 100 for CIFAR100, 102 for Oxford 102 Flower, and 1000 for ImageNet.

• LSC/DSC/MDSC (Likelihood Surprise Coverage/Distance-ratio Surprise Coverage/Mahalanobis Distance Surprise Coverage) [37–39] involve two crucial parameters: the bucket count (m) and the maximal SA value (U). However, we encounter challenges in tuning these hyperparameters, and the authors do not provide specific guidance regarding their selection. To simplify the presentation, we opt to use the bucket size ($T = \frac{U}{M}$) instead and report the covered buckets directly, rather than the ratios of covered buckets as originally proposed in the paper.

• CAC (Causal-Aware Coverage) [34], captures the causal relations of neurons, which are formed over neuron edges, and performs statistical independence tests to decide the causal relations derived from DNN edges. To incorporate a discrete updatable independence test method called $\mathcal{X}^2$-test, CAC groups continuous neuron values into K splits, and we set $K = 8$ which is consistent with the settings used in [34].

• NLC (NeuraL Coverage) [90], also known as "neural coverage", is distinct from other criteria as it emphasizes the overall activity of neuron groups within each layer and takes into account the relationships between neurons. Notably, NLC does not require any parameters for its calculation.

Moreover, to mitigate the computational cost associated with determining the output ranges of each neuron, we randomly selected 1,000 training samples to estimate the neuron output ranges for KMNC/NBC/SNAC. This sampling approach helps to reduce the computational complexity while still providing a representative coverage estimation. Notably, all coverage criteria will utilize the mutation rules described in Section 5.2 for generating mutants.

5.4 Evaluation Metrics

We anticipate that NSGen could enhance testing adequacy. To verify this, we utilized the following metrics [30, 90]:

1) *Triggered Fault*: A triggered fault occurs when a test input leads to a fault in the DNN, causing incorrect predictions. The **number of triggered faults** (denotes as #Faults) serves as a direct evaluation metric [90], reflecting the count of distinct errors or abnormal behaviors discovered during the fuzzing procedure.

2) *Fault Detection Rate*: Also, it is important to measure the **rate of triggered faults** (denotes as RFT) [27]. Larger RFT means that the generated test inputs have a tendency to challenge and uncover potential faults within

the DNN. Moreover, a higher RFT within a given time period means higher efficiency in revealing faults. RFT is calculated by dividing the number of triggered faults by the total number of generated test inputs. The formula for RFT is:

$$RFT = \frac{\#Faults}{\#Generated\ Test\ Inputs} \tag{13}$$

3) *Time Cost*: Similar to the RFT, a higher **fault-revealing efficiency** (denoted as FRE) signifies the criterion's capacity to efficiently uncover faults within a given time frame. FRE is computed by dividing the total time spent on a criterion by the number of triggered faults, yielding the following formula:

$$FRE = \frac{Total\ Time\ Cost}{\#Faults} \tag{14}$$

4) *Diversity*: A higher diversity of erroneous behaviors indicates a broader vulnerability surface of DNN models [81, 90]. This diversity is measured using the **number of covered classes** (denotes as #Classes) in a collection of fault-triggering images [90]. In cases where the number of covered classes is equal, we further assess the skew of the output class distribution using Pielou's evenness score [61]. This score is a theoretically grounded measure of biodiversity derived from normalized Shannon's entropy, scaled to a range between 0 and 1 by dividing the entropy of the output distribution by the maximum entropy given the total number of classes. A high evenness score signifies high impartiality or low bias. The **output impartiality** (denotes as OI) metric for a test suite T with $|C|$ possible classes is defined as follows [30]:

$$OI(T) = \frac{\sum_{t \in c} P_{t=c} \log P_{t=c}}{\frac{1}{|C|} \log \frac{1}{|C|}} \tag{15}$$

Here, $|C|$ represents the number of classes, $P_{t=c}$ denotes the percentage of test inputs t predicted to belong to class c, and a higher entropy value indicates higher diversity.

6 EVALUATION

We primarily study the following research questions (RQs).

- **RQ1:** To what extent does NSGen outperform baseline methods in terms of both effectiveness and efficiency in detecting faults?
- **RQ2:** To what extent does NSGen reveal the diversity of erroneous behaviors?
- **RQ3:** How does the choice of different text encoders affect the effectiveness of NSGen?
- **RQ4:** What are the effects of different hyperparameter configurations on NSGen?

6.1 RQ1: Effectiveness and Efficiency of NSGen

Setup: We investigate the effectiveness and efficiency of using existing coverage criteria and NSGen as feedback guidance for input mutations. For each baseline, we summarize its hyperparameter settings in Section 5.3. Additionally, we employ a systematic approach to set distinct similarity thresholds for NSGen, denoted as τ, across various datasets. Specifically, we set τ to 0.81 for CIFAR10, 0.71 for CIFAR100, 0.87 for Oxford 102 Flower, and 0.72 for ImageNet (*c.f.* Section 4.3.4). The fuzzing procedure commences by constructing an initial seed pool. During the fuzzing cycle, seeds drawn from this pool are subjected to mutations. Each seed undergoes transformations across three mutation themes: pixel-level, affine type, and style transfer, resulting in a total of 95 mutation styles (*cf.* Section 5.2). Ensuring that the mutated images retain meaningful content is essential; hence, we configure α to 0.2 and β to 0.4 as per established settings. After processing, for the baseline methods, the DNN evaluates the mutated seed and collects coverage information. If the coverage increases, it will be re-added to the pool for subsequent use. In contrast, for NSGen, the retention of mutated seeds is determined using the

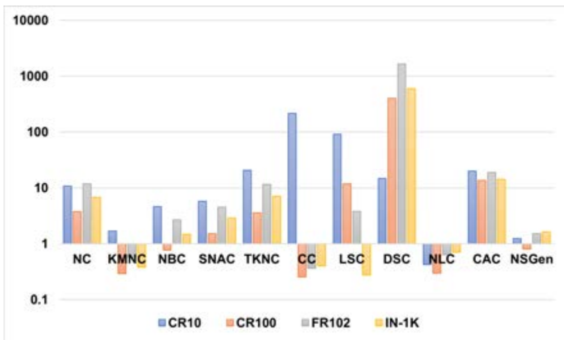
Table 3. Fuzzing results. Faults, Outputs, and RFT denote triggered faults, fuzzing outputs, and rate of triggered faults, respectively. Best assessments are marked .

Models	Criteria	CIFAR10			CIFAR100			Oxford 102 Flower			ImageNet		
		#Faults/#Outputs	RFT	Coverage	#Faults/#Outputs	RFT	Coverage	#Faults/#Outputs	RFT	Coverage	#Faults/#Outputs	RFT	Coverage
ResNet	NC	1252/1897	0.6600	0.6327	3012/3591	0.8388	0.8133	1813/2635	0.6880	0.4235	3160/4041	0.7820	0.5833
	KMNC	5157/9889	0.5215	0.9717	7369/10000	0.7369	0.9511	5636/10000	0.5636	0.9678	6877/10000	0.6877	0.9415
	NBC	2297/4918	0.4671	0.8460	6332/8501	0.7449	0.9053	4282/7236	0.5918	0.9461	6008/8554	0.7024	0.8231
	SNAC	1656/3795	0.4364	0.8516	4408/6179	0.7134	0.9260	3106/5410	0.5741	0.9560	4496/6581	0.6832	0.8689
	TKNC	1042/1608	0.6480	0.2534	3482/4159	0.8372	0.5330	1858/2803	0.6629	0.3411	3032/3882	0.7810	0.4983
	TKNP	0/1	0	1.0	0/1	0	1.0	0/1	0	1.0	0/1	0	1.0
	CC	55/84	0.6548	84.0	9320/9999	0.9321	9999.0	6863/10000	0.6863	10024.0	7824/10000	0.7824	10001.0
	LSC	101/117	0.8632	117.0	814/1035	0.7865	1035.0	3662/4937	0.7417	4937.0	6818/10000	0.6818	10000.0
	DSC	877/888	0.9876	888.0	38/45	0.8444	45.0	13/26	0.5000	26.0	36/44	0.8182	44.0
	MDSC	5146/5733	0.8976	5733.0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	CAC	1080/2775	0.3892	0.5580	1587/2611	0.6078	0.3620	1138/2526	0.4505	0.3241	1504/2504	0.6006	0.1820
	NLC	5010/10000	0.5010	211.48	7505/10000	0.7505	9639.2	5915/9966	0.5935	174393.9	6651/9980	0.6664	1820913.3
	NSGen	8075 /10000	0.8075	10.0	9653 /9999	0.9654	9.9990	8394 /9990	0.8402	9.9900	9815 /9935	0.9879	9.9350
VGG	NC	765/1088	0.7031	0.4902	478/581	0.8227	0.7687	696/935	0.7444	0.4344	1953/2386	0.8185	0.8566
	KMNC	4017/8481	0.4736	0.9690	6577/9366	0.7022	0.9324	5741/9489	0.6050	0.9411	7110/9999	0.7111	0.8365
	NBC	748/2202	0.3397	0.6024	1765/2815	0.6270	0.7709	1364/2762	0.4938	0.8298	2500/4294	0.5822	0.5034
	SNAC	646/1757	0.3677	0.9017	1358/2142	0.6340	0.8908	1007/2047	0.4919	0.7894	2114/3716	0.5689	0.7007
	TKNC	837/1230	0.6805	0.2960	882/1080	0.8167	0.8642	385/621	0.6200	0.2878	2245/2791	0.8044	0.8105
	TKNP	1/1	1	1.0	0/1	0	1.0	1/1	1	1.0	0/1	0	1.0
	CC	106/167	0.6347	224.0	9104/10000	0.9104	12628.0	6114/10000	0.6114	25050.0	8227/10000	0.8227	17996.0
	LSC	525/569	0.9227	569.0	5469/6402	0.8543	6402.0	5994/10000	0.5994	10000.0	7284/10000	0.7284	10000.0
	DSC	1381/1393	0.9914	1393.0	135/141	0.9574	141.0	40/50	0.8000	50.0	48/56	0.8571	56.0
	MDSC	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	CAC	2628/5293	0.4965	0.8907	7199/10000	0.7199	0.6246	N/A	N/A	N/A	N/A	N/A	N/A
	NLC	5344/9979	0.5355	132625.7	7341/9967	0.7365	198764.7	5988/9668	0.6194	421299968.0	6369/9394	0.6780	11601800.0
	NSGen	8393 /9995	0.8397	9.9950	9644 /9997	0.9647	9.9970	7872 /9175	0.8580	9.1750	9665 /10000	0.9665	10.0
MobileNet	NC	254/464	0.5474	0.6502	327/457	0.7155	0.7368	610/901	0.6770	0.8308	620/791	0.7838	0.8713
	KMNC	5243/9916	0.5287	0.8974	7269/10000	0.7269	0.7388	5770/10000	0.5770	0.9691	7620/10000	0.7620	0.9477
	NBC	1962/4557	0.4305	0.7551	3988/5890	0.6771	0.6767	3233/5470	0.5910	0.9255	5124/6958	0.7364	0.8147
	SNAC	1359/3291	0.4129	0.7524	2637/4011	0.6574	0.6904	2220/4020	0.5522	0.9462	3503/5041	0.6949	0.8555
	TKNC	300/546	0.5495	0.2698	600/810	0.7407	0.3770	1250/2055	0.6083	0.6115	1242/1640	0.7573	0.6875
	TKNP	0/1	0	1.0	0/1	0	1.0	0/1	0	1.0	0/1	0	1.0
	CC	25/53	0.4717	53.0	8427/10000	0.8427	10000.0	6716/10000	0.6716	10047.0	8070/10000	0.8070	10002.0
	LSC	77/82	0.939	82.0	1830/2365	0.7738	2365.0	4128/5852	0.7054	5852.0	7649/10000	0.7649	10000.0
	DSC	1408/1418	0.9929	1418.0	21/29	0.7241	29.0	21/36	0.5833	36.0	42/46	0.9130	46.0
	MDSC	2830/3460	0.8179	3460.0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	CAC	1731/3803	0.4552	0.2842	2662/4066	0.6547	0.2037	1791/3926	0.4562	0.5747	2677/4009	0.6677	0.3500
	NLC	5418/10000	0.5418	120.39	7493/10000	0.7493	25093.2	6031/9874	0.6108	1971819.6	7302/9801	0.7450	6650560.5
	NSGen	8671 /9997	0.8674	9.9970	9654 /10000	0.9654	10.0	8695 /9981	0.8712	9.9810	8867 /8993	0.9860	8.9930

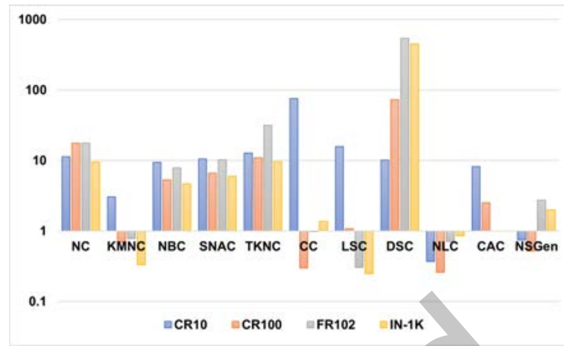
procedure outlined in Algorithm 1. The fuzzing process is initiated with 1,000 randomly selected inputs from the test dataset as fuzzing seeds[7]. To ensure practicality, we establish a termination condition for fuzzing, halting it when it reaches 10,000 iterations or exceeds a 6-hour limit, aligning with the setting utilized in [90]. To mitigate the non-trivial bias in this random process, we repeat each experiment five times with different randomly selected sets of seeds and then calculate the average results across these iterations. We evaluate performance using three key metrics: the number of triggered faults (#Faults), the rate of triggered faults (RFT), and fault-revealing efficiency (FRE). #Faults and RFT help us assess the effectiveness of each guidance criterion in directing the fuzzing framework to generate fault-triggering images, while FRE measures the efficiency of generating such images.

6.1.1 Effectiveness. Table 3 shows the #Faults and RFT of NSGen and baselines with different DNN and dataset configurations. We highlight testing criteria with the best results. It is worth noting that completing all the criteria, including LSC, DSC, and MDSC, poses practical challenges. These criteria require the entire training set for initialization, and for each test input, LSC and DSC involve iterating over all neuron output traces generated using the training data, making their execution time unacceptable, especially for ImageNet. Moreover, MDSC relies on storing a class conditional covariance matrix to represent the training data, but due to the number of classes in CIFAR100, Oxford 102 Flower, and ImageNet, it demands an excessive amount of GPU memory,

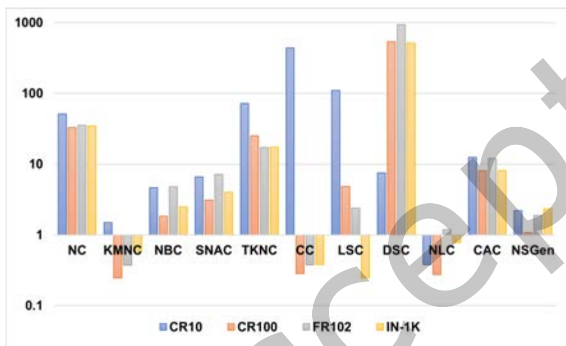
[7]Apart from the necessary implementation and hyperparameter settings for NSGen, we adhere to and strictly follow the experimental framework adopted by [90]



(a) FRE (seconds) on testing ResNet50



(b) FRE (seconds) on testing VGG16_BN



(c) FRE (seconds) on testing MobileNet_v2

Fig. 5. Comparison in terms of FRE (y-axis shows the average time to generate a fault-revealing test input)

rendering its evaluation infeasible. As a result, we do not report the results for MDSC on ResNet50, VGG16_BN, and MobileNet_v2 trained on these three datasets.

Among the compared criteria, images mutated under the guidance of NSGen exhibit the highest #Faults compared to other criteria regardless of the datasets and the models, particularly NC and NBC. For example, when compared to the state-of-the-art coverage criterion NLC, NSGen leads to a remarkable increase in #Faults, ranging from 21.4% to 61.2%. This difference in performance can be attributed to the way these criteria categorize neurons as activated. NC and NBC consider a neuron activated if its output exceeds a certain threshold or falls outside a specified range. However, DNNs often incorporate normalization techniques, causing the outputs of neurons to concentrate within specific regions. As a result, without access to fine-grained feedback such as gradients, mutated images may struggle to switch an inactive neuron to an active state. In the case of NLC, it may be due to the difficulty of using layers as the unit of calculation to capture subtle differences between various test inputs.

Additionally, NSGen consistently demonstrates strong performance in terms of RFT across different datasets and models. For example, RFT values are consistently above 0.8, and NSGen excels on more complex datasets like CIFAR100, Oxford 102 Flower, and ImageNet. The complexity of these datasets allows neurons to focus on diverse fine-grained features, effectively distinguishing between neurons' semantic paths. On the CIFAR10 dataset, the RFT of DSC approaches 1.0. This is because DSC requires generating test inputs that elicit "surprise" behavior from the DNN, leading to test inputs that focus on uncommon or rare scenarios, resulting in fewer inputs being generated, but with a higher probability of exposing DNN faults. Unfortunately, CAC, the first coverage criterion rooted in a causal perspective, falls short when dealing with complex and high-dimensional systems, such as DNNs trained on extensive datasets. Therefore, in this experimental setup, generating numerous test inputs becomes impractical for CAC. On average, NSGen outperforms CAC by generating 387.98% more #Faults. TKNC and CC also exhibit decent performance, as they partly capture patterns and clusters that reflect neuron interdependencies.

At the same time, the coverage results (DPD is the coverage achieved by NSGen) in Table 3 provide further substantiate findings from previous studies [30, 47, 69, 87, 88], highlighting that certain neuron coverage criteria are capable of achieving full coverage with a relatively small number of test inputs. This observation aligns with our motivation to explore more efficient testing methodologies that do not aim for maximum coverage but enhance the actual quality of the model. For instance, in Table 3, SNAC achieved coverage of 0.9462 on the MobileNet_v2/Flowers102 using only 4,020 test inputs. In contrast, KMNC required a full 10,000 test inputs to reach a coverage of 0.9691. This disparity not only illustrates the varying efficiencies of different coverage metrics but also supports our assertion that increasing the coverage does not invariably correlate with improved model quality.

6.1.2 Efficiency. Based on the experimental data generated in Section 6.1.1, we further measure the fault-revealing efficiency (FRE) for generating a fault-revealing image during the fuzzing process, whose results are shown in Figures 5(a), 5(b), and 5(c). Among them, the part below the x-axis indicates that the criterion takes less than 1 second to generate a fault-revealing image, and the longer the column length, the shorter the time; conversely, the part above the x-axis indicates that the generation time is greater than 1 second, and the longer the column length, the longer the time. From these figures, NSGen performs competitive fault-revealing efficiency compared to other criteria regardless of the datasets and models. For example, in Figure 5(a), on average, it takes 1.302 seconds to generate a fault-revealing image, ranking 3/12. On the CIFAR100 dataset, NSGen performs even better, generating a fault-revealing image in just 0.8 seconds. NLC stands out as the most efficient criterion, as its fault-revealing test input generation time is consistently below 1 second for all datasets, indicated by the columns residing below the x-axis. However, when it comes to effectiveness, NSGen outperforms NLC by generating an average of 2,585 more fault-revealing images. We also find that while DSC exhibits a higher RFT value than NSGen on the CIFAR10 dataset (*c.f.* Section 6.1.1), it's essential to note that, in terms of efficiency, generating a fault-revealing image with DSC takes, on average, 672.48% more time than using NSGen. In summary, NSGen's FRE can rank in the top 3 out of 12 criteria across all datasets on average.

Overall, the guidance of neuron semantic differences in NSGen largely improves the effectiveness and efficiency of test input generation, which is also the reason why most criteria underperform NSGen.

Answer to RQ1: Under the same time constraints, NSGen outperforms the other 12 coverage criteria in terms of the number of triggering DNN failures, exhibiting a remarkable increase in the number of triggered faults by 21.4% to 61.2% compared to the state-of-the-art coverage-guided fuzzing criterion. Moreover, partial neural coverage criteria can achieve full coverage with relatively few test inputs. Additionally, NSGen's time efficiency can rank in the top 3 out of 12 criteria.

6.2 RQ2: Diversity of Erroneous Behaviors.

Setup: As described in [6, 30, 82, 90], the generated test inputs should be diverse in terms of the outputs. Existing work [69] has demonstrated that neuron coverage can be improved with a few samples. However, test inputs containing only one class of samples (e.g., bird) are not enough to comprehensively reveal diverse faults of the DNN (e.g., the errors in other classes). Additionally, as mentioned in [82], coverage criteria that are sensitive to output diversity can guide the testing tools to generate more diverse test inputs belonging to different classes,

Table 4. Diversity results. Classes and OI denote covered classes and output impartiality, respectively. Best assessments are marked .

Models	Criteria	CIFAR100		Oxford 102 Flower		ImageNet		CIFAR10	
		#Classes	OI	#Classes	OI	#Classes	OI	#Classes	OI
ResNet	NC	99	—	100	—	647	—	10	0.9737
	KMNC	99	—	101	—	709	—	10	0.9428
	NBC	99	—	101	—	750	—	10	0.9779
	SNAC	100	0.9127	100	—	723	—	10	0.9818
	TKNC	99	—	100	—	723	—	10	0.9723
	TKNP	1	—	1	—	1	—	1	—
	CC	99	—	102	0.8663	720	—	10	0.9869
	LSC	93	—	102	0.8476	694	—	10	0.8477
	DSC	14	—	21	—	30	—	10	0.7813
	MDSC	N/A	N/A	N/A	N/A	N/A	N/A	10	0.8399
	CAC	95	—	90	—	433	—	10	0.9610
	NLC	100	0.8764	101	—	717	—	10	0.9226
	NSGen	100	0.9195	102	0.8618*	751	—	10	0.8515*
VGG	NC	90	—	93	—	582	—	10	0.9707
	KMNC	99	—	102	0.9374	699	—	10	0.9640
	NBC	100	0.8816	101	—	584	—	10	0.9787
	SNAC	97	—	100	—	583	—	10	0.9790
	TKNC	95	—	94	—	619	—	10	0.9727
	TKNP	1	—	1	—	1	—	10	0.9640
	CC	99	—	102	0.9204	695	—	10	0.9941
	LSC	99	—	101	—	671	—	10	0.7581
	DSC	17	—	31	—	25	—	10	0.8374
	MDSC	N/A	N/A	N/A	N/A	N/A	N/A	10	0.9613
	CAC	99	—	N/A	N/A	N/A	N/A	10	0.9385
	NLC	100	0.8108	102	0.9342	610	—	10	0.8693
	NSGen	100	0.8871	102	0.8991*	702	—	10	0.8782*
MobileNet	NC	86	—	95	—	334	—	10	0.9762
	KMNC	99	—	102	0.8993	767	—	10	0.9400
	NBC	99	—	102	0.8964	767	—	10	0.9762
	SNAC	99	—	101	—	705	—	10	0.9811
	TKNC	96	—	102	0.8904	509	—	10	0.9809
	TKNP	1	—	0	—	1	—	0	—
	CC	100	0.8436	102	0.8399	770	—	10	0.9741
	LSC	97	—	102	0.8451	764	—	10	0.8525
	DSC	14	—	28	—	27	—	9	0.6573
	MDSC	N/A	N/A	N/A	N/A	N/A	N/A	10	0.9613
	CAC	98	—	102	0.9346	599	—	10	0.9385
	NLC	98	—	102	0.8941	747	—	10	0.9061
	NSGen	100	0.8466	102	0.8486*	776	—	10	0.8540*

*Higher OI is better if two #Classes are equal, otherwise, higher #Classes is better.

while insensitive coverage criteria may generate biased test inputs. Hence, we evaluate the output diversity of fuzzing guided by NSGen as well as other coverage criteria in this RQ. We assess the diversity using two key metrics: the number of covered classes (#Classes) and the output impartiality (OI). It's important to note that our primary focus is on #Classes. Therefore, in Table 4, our initial calculation centers around #Classes. If there's a tie in #Classes, we proceed to analyze OI. In the cases where the #Classes differs, we fill the table with short horizontal lines. For our diversity analysis, we continue to utilize the experimental data derived from RQ1 (*c.f.* Section 6.1).

Result: The results presented in Table 4 reveal that when NSGen is employed as the guidance mechanism for fuzzing, it produces fault-revealing images covering all categories in the CIFAR10, CIFAR100, and Oxford 102

Flower datasets. Moreover, on the ImageNet dataset, NSGen's fault-revealing images encompass a majority of the classes, surpassing the state-of-the-art method, NLC, by an average of 51 classes. For example, when considering the VGG/ImageNet combination, NSGen surpasses NLC by 15.08% in coverage classes. When the #Classes is the same, NSGen's performance, as indicated by the OI, demonstrates competitiveness with other standards. However, for the CIFAR10 dataset, most coverage criteria manage to cover all 10 classes, and in terms of OI, NSGen slightly lags behind some coverage criteria. One possible explanation for this observation is that the CIFAR10 dataset may not possess sufficient richness and complexity compared to the more extensive ImageNet dataset. As a result, the differences in performance between NSGen and other criteria are less pronounced on CIFAR10.

We also find that NSGen's diversity performance improves as the dataset complexity increases. Particularly when applied to larger and more intricate datasets like CIFAR100 and ImageNet, this underscores NSGen's potential to identify a broader range of flaws in real-world critical DNN systems.

Answer to RQ2: In summary, NSGen outperforms the remaining 12 coverage criteria in terms of output diversity, achieving full class coverage on CIFAR10, CIFAR100, and Oxford 102 Flower datasets while surpassing the state-of-the-art criterion on ImageNet by an impressive average of 51 additional classes.

6.3 RQ3: Impact of Different Text Encoders

Setup: NSGen utilizes CLIP (*c.f.* Section 4.3.3) to vectorize the templates of the original and mutated images to measure the similarity between them. Therefore, to answer RQ3, we investigate how various language models and word embedding methods affect NSGen's performance. We consider three language models: BERT [23], Roberta-L [48], and GPT2 [63]. These models are chosen because they are pre-trained on extensive, unlabeled text data, allowing them to dynamically capture rich semantic information from text. In contrast, the word embedding methods we explore are FastText [35] and GloVe [60], which provide fixed representations of text. We assess the influence of different encoders on NSGen's performance using three key metrics: the number of triggered faults (#Faults), the number of covered classes (#Classes), and output impartiality (OI). The experimental setup remains consistent with that of RQ1 and RQ2, with the primary change being the substitution of the encoder in NSGen.

Result: The findings are summarized in Table 5. Overall, our method's integration of CLIP as the text vectorization component has shown superior performance, especially for more complex datasets such as CIFAR100, Oxford 102 Flower, and ImageNet. For example, consider the performance of NSGen on the ImageNet dataset. NSGen excels by generating an average of 8640 more fault-triggering images compared to BERT and 3476 more than FastText. In terms of diversity, NSGen covers 497 more classes on average compared to BERT and 134 more classes compared to FastText. These results underscore CLIP's effectiveness in capturing and leveraging relevant information from both textual and visual domains. It is important to note that when CLIP is used as the text vectorization component on the CIFAR10 dataset, the final guidance results fall slightly short compared to traditional word embedding methods like FastText and GloVe. This discrepancy can be attributed to the limited diversity in the available neuron descriptions associated with the CIFAR10 dataset. As the CIFAR10 dataset is relatively simple in terms of image classification tasks, it may not capture the comprehensive scene information required for neurons to express semantic features adequately.

Indeed, it is intriguing that despite their extensive pretraining, large language models demonstrate performance comparable to simple word embeddings. This suggests that the full potential of pre-trained language context has not been fully harnessed for neuron semantic guidance. This observation opens up an exciting avenue for future research, where further exploration and investigation are warranted to fully utilize the untapped potential of large language models in neuron semantic-guided tasks.

Table 5. Impact of different text encoders. Faults, Outputs, and OI denote triggered faults, fuzzing outputs, and output impartiality, respectively. CR10, CR100, FR102 and IN-1K denote CIFAR10, CIFAR100, Oxford 102 Flower and ImageNet-1k dataset, respectively. Best assessments are marked.

Models	Dataset	ResNet			VGG			MobileNet		
		#Faults/#Outputs	#Classes	OI	#Faults/#Outputs	#Classes	OI	#Faults/#Outputs	#Classes	OI
(a) Language Models										
+ BERT [23]	CR10	3490/3449	10	0.86	9009/9915	10	0.78	1297/1417	10	0.81
	CR100	5923/6323	98	—	6041/6493	98	—	697/768	80	—
	FR102	670/828	84	—	67/86	39	—	124/155	52	—
	IN-1K	796/812	311	—	1569/1607	376	—	62/62	49	—
+ Roberta-L [48]	CR10	1882/2069	10	0.86	7865/8636	10	0.80	822/879	10	0.79
	CR100	4820/5077	99	—	9150/9568	100	0.88	3896/4076	100	0.84
	FR102	344/401	75	—	53/58	27	—	301/362	66	—
	IN-1K	559/569	243	—	1113/1136	321	—	31/31	28	—
+ GPT2 [63]	CR10	4947/5543	10	0.87	8880/9952	10	0.77	1944/2150	10	0.79
	CR100	7281/10000	98	—	7287/10000	98	—	7381/10000	100	—
	FR102	5670/10000	102	0.91	6202/10000	102	0.92	5624/10000	102	0.89
	IN-1K	678/687	268	—	1677/1731	387	—	30/30	25	—
(b) Word Embeddings										
+ FastText [35]	CR10	8369/10000	10	0.84	8112/10000	10	0.77	8748/9990	10	0.83
	CR100	7933/10000	99	—	7866/10000	99	—	8027/10000	99	—
	FR102	6605/10000	102	0.87	4787/7052	101	—	6850/10000	102	0.85
	IN-1K	7550/8060	703	—	8726/9421	691	—	1641/1660	433	—
+ GloVe [60]	CR10	6809/7579	10	0.87	9018/9943	10	0.79	2415/2626	10	0.81
	CR100	9431/9753	100	0.90	9443/9930	99	—	1981/2075	95	—
	FR102	3854/4624	101	—	744/1100	99	—	551/721	82	—
	IN-1K	1678/1721	424	—	3011/3120	507	—	125/125	76	—
(c) Language-Image Models										
+ CLIP [62]	CR10	8075/10000	10	0.85*	8393/9995	10	0.80	8671/9997	10	0.86
	CR100	9653/9999	100	0.92	9644/9997	100	0.89	9654/10000	100	0.85
	FR102	8394/9990	102	0.86*	7872/9175	102	0.90*	8695/9981	102	0.85*
	IN-1K	9815/9935	751	—	9665/10000	702	—	8867/8993	776	—

*Higher OI is better if two #Classes are equal, otherwise, higher #Classes is better.

Answer to RQ3: NSGen utilizes CLIP as the default text encoder. The performance comparison with alternative three language model architectures and two word embedding approaches confirms the validity of our selection.

6.4 RQ4: Impact of NSGen Configurations

In this section, we study hyperparameters that can affect the effectiveness of NSGen. We examine five key hyperparameters: Number of Top-Activating Images, LSTM Configurations, Number of Selected Layers, Number of Class Labels, and Threshold Stability Across Different Sample Sets.

6.4.1 Number of Top-Activating Images. Top-activating images play a crucial role in representing specific visual features or patterns recognized by neurons. NSGen aggregates common features from these images and generates natural language descriptions for neurons (*c.f.* Section 4.2.1). A significant challenge arises when there are too many top-activating images, making it difficult to extract shared features and reducing the quality of the natural language descriptions [4]. This section explores how different quantities of top-activating images affect the quality of the generated descriptions. Based on previous research [8, 57], we aimed to create natural language descriptions for neurons in the final layer of the model to evaluate their quality. This is because it is difficult to establish a definitive ground truth for natural language descriptions of neurons, while the output layer neurons' ground truth is explicit and aligned with the class labels. To evaluate the alignment between the true class

Table 6. Cosine similarity of natural descriptions generated on different numbers of top-activating images. The higher the similarity, the better.

Top-k Images	ImageNet/ResNet50		Flowers102/MobileNet_v2		CIFAR10/VGG16_BN		CIFAR100/ResNet50	
	CLIP cos	mpnet cos	CLIP cos	mpnet cos	CLIP cos	mpnet cos	CLIP cos	mpnet cos
k=1	0.7163	0.2518	0.6587	0.3388	**0.8330**	0.3426	0.7842	0.2666
k=5	0.7222	0.2674	0.6675	**0.3670**	0.8213	0.3296	0.7788	0.2551
k=10	0.7237	**0.2704**	0.6660	0.3653	0.8184	0.3521	0.7837	0.2613
k=15	**0.7241**	0.2701	**0.6675**	0.3659	0.8286	**0.3873**	**0.7856**	0.2546
k=20	0.7231	0.2703	0.6670	0.3661	0.8242	0.3717	0.7847	**0.2696**

Table 7. Cosine similarity of natural descriptions generated on different configurations of LSTM. The higher the similarity, the better.

LSTM Config.	ImageNet/ResNet50		Flowers102/MobileNet_v2		CIFAR10/VGG16_BN		CIFAR100/ResNet50	
	CLIP cos	mpnet cos	CLIP cos	mpnet cos	CLIP cos	mpnet cos	CLIP cos	mpnet cos
IES=128, HS=512	0.7241	0.2701	**0.6675**	**0.3659**	0.8286	**0.3873**	0.7856	0.2546
IES=128, HS=128	**0.7285**	0.2650	0.6670	0.3620	0.8018	0.2469	0.7856	0.2369
IES=128, HS=256	0.7266	0.2579	0.6675	0.3339	0.8340	0.3676	**0.7949**	**0.2717**
IES=128, HS=1024	0.7144	0.2145	0.6616	0.3407	0.8286	0.2508	0.7905	0.2297
IES=64, HS=512	0.7256	0.2718	0.6670	0.3618	0.8232	0.3315	0.7837	0.2213
IES=256, HS=512	0.7207	**0.2754**	0.6660	0.3623	0.7832	0.2790	0.7827	0.2154
IES=512, HS=512	0.7251	0.2542	0.6646	0.3593	**0.8408**	0.3376	0.7925	0.2201

names of neurons and the generated natural language descriptions, we measure the cosine similarity between the neuron's true class name and the sentence embedding space of the natural language description generated by the method [8, 57]. For embeddings, we use two different encoders: the CLIP ViT-B/32 text encoder (denoted CLIP cos) and the all-mpnet-base-v2 sentence encoder (denoted mpnet cos). The experimental results (see Table 6) illustrate a trend where the quality of the natural language descriptions of neurons, generated by varying the count of top-activating images, initially improves but subsequently diminishes with an increase in the number of images. Specifically, in Flowers102/MobileNet_v2 and CIFAR10/VGG16_BN, the description quality reaches its peak when the number of top-activating images is between 10 to 15 and then starts to decline. Thus, choosing the right number of top-activating images is essential to precisely depict and explain the activation patterns of neurons. Using too few top-activating images may result in incomplete coverage, while using too many may introduce noise and compromise the quality of the description.

6.4.2 LSTM Configurations. NSGen uses an LSTM-based decoder with an input embedding size (IES) of 128 and a hidden size (HS) of 512 to generate natural language descriptions of neurons (*c.f.* Section 4.2.2). This section analyzes the effect of different LSTM settings on the quality of these descriptions. The impact is examined by keeping one setting constant while varying the other, following the methodologies outlined in Section 6.4.1. The experimental results are shown in Table 7. Regarding input embedding size, increasing it appropriately can enhance the model's ability to capture nuanced semantics. For example, in ImageNet/ResNet50 and CIFAR10/VGG16_BN, an increase in input embedding size is associated with improved quality in the generated descriptions. This improvement is likely a result of the model's enhanced ability to represent a wide range of vocabulary and complex syntactic structures within embeddings. However, it is important to recognize that there is a limit beyond which further expansion does not lead to significant improvements in generation quality. This observation indicates that a larger embedding size provides a more extensive representation space, but also requires more data to effectively assimilate these representations without overfitting. Regarding the hidden state size, increasing the hidden size significantly improves the model's ability to generate complex sentence structures, which is particularly important for describing highly abstract visual features. In CIFAR10/VGG16_BN and CIFAR100/ResNet50, a relatively larger hidden size results in significant improvements in sentence embedding similarity (i.e., mpnet cos). However, excessively large hidden layers increase the model's parameter count, which in turn increases the risk of overfitting. A comparison of different combinations of input embedding and hidden sizes showed that an intermediate input embedding size (e.g., 128) combined with a relatively large hidden size (e.g., 512) achieves optimal performance. This configuration maintains the model's descriptive generation capabilities while also addressing the overfitting issues associated with excessive parameters.

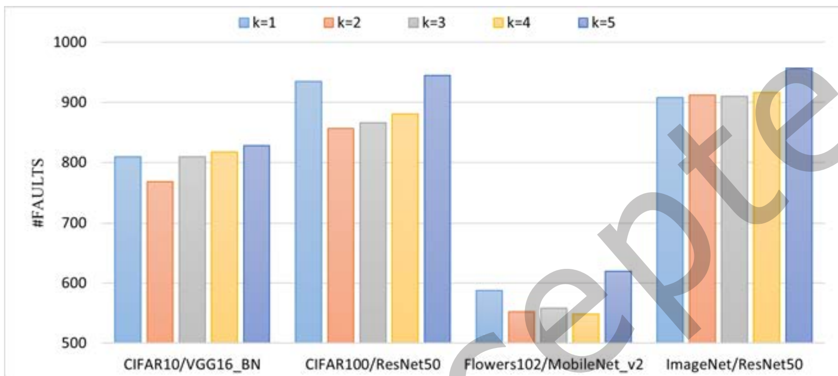


Fig. 6. The performance of NSGen under different layer numbers.

6.4.3 Number of Selected Layers. DNNs are composed of multiple layers, each of which contributes differently to the network's ability to process and interpret input data. Layers closer to the input tend to capture more general and broadly applicable features, while deeper layers focus on more abstract and specific features [32]. NSGen selects neurons from specific k layers selected in the DNN to construct semantic decision paths. As outlined in Section 4.3.1, the values of k for ResNet50, VGG16_BN, and MobileNet_v2 are set to 5, 5, and 10, respectively. To assess the impact of different numbers of model layers on NSGen, we adjust the number of layers included in the neuron selection phase in accordance with the model's layer execution order. Neurons from these selected layers are then utilized to guide the semantic decision-making process, adhering to the NSGen framework. The primary metric for this evaluation is the number of triggered faults (#Faults), which serves to determine the impact of different k values on NSGen's effectiveness. Experimental results, as depicted in Figure 6, clearly demonstrate that the value of k significantly influences the #Faults across different networks. For example, when $k = 5$, the #Faults is increased by 1.07% to 13.14% in comparison to other values of k. Overall, as the k value increases, the #Faults have been increasing, but will gradually stabilize. This indicates that increasing the number of layers can provide richer semantic information, allowing NSGen to distinguish normal inputs from mutated inputs from the perspective of decision path discrepancy (DPD). Notably, the NSGen variant with $k = 1$ triggers more faults than variants with more layers. One potential explanation is that low-level, general features captured at lower levels play a critical role in early fault detection because they serve as the basis for complex decisions at higher levels.

6.4.4 Number of Class Labels. Class labels are vital in the construction of templates (see Section 4.3.2), and their number directly influences the performance of NSGen. In our previous experiments, we employed top-1 class labels for CIFAR10, CIFAR100, and Oxford 102 Flower datasets, and top-3 class labels for the ImageNet dataset.

Table 8. The performance of NSGen under different class label numbers.

Dataset	Model	Top-k (class labels)	#Faults/#Outputs	#Classes
CIFAR10	MobileNet	top-1	**8671/9997**	**10**
		top-2	8594/9967	10
		top-3	4189/4810	10
ImageNet	ResNet	top-1	9641/9993	740
		top-2	9614/9972	730
		top-3	**9815/9935**	**751**
		top-4	7202/7356	690

This choice is supported by prior studies [64], which have shown that using an appropriate number of categories provides more information about the input, leading to the inclusion of richer visual concepts. By utilizing class labels effectively, NSGen can enhance its performance and produce more comprehensive and informative results. Our experiments have revealed that introducing an excessive number of class labels can lead to conflicting visual concepts, which in turn, negatively impacts the performance of NSGen. For example, when considering the CIFAR10 dataset, which contains 10 distinct categories, employing top-3 class labels results in a notable decline in NSGen's performance (see Table 8). Hence, it is essential to strike a balance in selecting the number of class labels to ensure the effective functioning of NSGen and achieve optimal results. The appropriate choice of class labels plays a crucial role in enhancing the performance and effectiveness of the NSGen, enabling it to detect more failures of DNNs.

Table 9. Thresholds on different datasets at different sampling rates (SR).

Dataset	SR=1%	SR=2%	SR=3%	SR=4%	SR=5%
CIFAR10	0.8097	0.8059	0.8042	0.8063	0.8142
CIFAR100	0.7073	0.7112	0.7167	0.7066	0.7125

6.4.5 Threshold Stability Across Different Sample Sets. The threshold τ plays a crucial role in determining whether to retain the generated mutated images, significantly impacting the performance of NSGen. As detailed in Section 4.3.4, NSGen samples a specific sample ratio (i.e., SR=1%) of the training set examples and traverses the mutation space defined by 95 predetermined mutation rules to establish the lower quartile of the similarity distribution as the threshold for the dataset. In this section, we examine the stability of these generated thresholds across different sampling rates. Due to the extensive time required for mutation processing on the ImageNet and Oxford 102 Flower datasets, our experiments are confined to the CIFAR10 and CIFAR100 datasets. Specifically, we incrementally extract 1%, 2%, 3%, 4%, and 5% of the data from each category in the CIFAR10/CIFAR100 training set and determine the final threshold following the methodology outlined in Section 4.3.4. The results are presented in Table 9, where the gray columns are the final thresholds we determined. We observed that the specific value of τ may vary slightly with different random samples, typically within a margin of ±0.01. Employing the lower quartile method minimizes this variation, ensuring a consistent threshold that keeps the calculated τ-values within an acceptable range. This approach enhances the robustness of our experiments, confirming the reliability of our threshold-setting process under varied conditions.

Answer to RQ4: This section examines the impact of NSGen's configurations, revealing that: 1) Optimal descriptions with 10-15 top-activating images strike the perfect balance between detail and clarity. 2) An input embedding size of 128 and a hidden size of 512 in LSTM settings facilitate detailed and accurate descriptions. 3) For ResNet50, VGG16_BN, and MobileNet_v2, the number of selected layers is set to 5, 5, and 10, respectively, which optimizes NSGen's fault detection capabilities. 4) A balanced choice of class labels is crucial to maintaining the quality of NSGen outputs. 5) The threshold τ maintains stable performance across varying sample sizes in CIFAR10 and CIFAR100, with variations within a ±0.01 margin, ensuring NSGen's reliability.

7 DISCUSSION

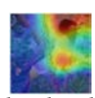
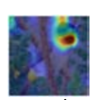
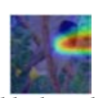
7.1 Interpretability of NSGen

Existing coverage criteria commonly share a limitation regarding the lack of interpretability of their test results. For example, the NC criterion oversimplifies the continuous output of neurons into binary states of activated or unactivated [90]. It can not provide a human-understandable decision path in analyzing the test results and accurately identify the neurons responsible for decision errors [81].

To address these shortcomings, NSGen takes a different approach by mapping the semantic information of neurons that significantly impact DNN's final prediction results into natural language descriptions. It organizes these descriptions in the order of layers, emulating DNN's layer-based decision transfer and forming a decision path based on neuron semantics.

As shown in Table 10, the class activation maps [96] illustrate the semantic features captured by neurons, with corresponding descriptions (***d***) generated by NSGen based on these semantic features (*c.f.* Section 4.2) labeled alongside. NSGen deconstructs the decision path that predicts "bulbu" for the original image input, revealing the underlying semantics that influences the model's decision. For instance, the presence of "stick" and "eye" enables the DNN to classify the image as an "animal on a tree," and subsequently, "beak" narrows the possible classes to "bird", leading to the final decision of "bulbul" based on above features. However, after mutating the original image, the prediction result becomes "indri." This discrepancy arises because subsequent neurons primarily capture features related to the tail and black-and-white stripes, leading to an incorrect model decision. By providing a detailed and interpretable decision path based on neuron semantics, NSGen enhances the understanding of DNN's decision-making process, making it valuable for model analysis and improvement.

Table 10. DNN decision (identified by NSGen) for a bulbul image and the mutated one with blur.

Input	**Neuron-semantic Features Identified by NSGen**							**Pred.**
 original	 stick	→	 eye	→	 beak	→	 head and stripes	bulbul
 mutated	 stick	→	 tail, stripes	→	 ear, stripes	→	 black-and-white stripes	indri

7.2 Contribution of Main Components

We delved into the individual contributions of its core components: Critical Neuron Selection (CNS) and Natural Language Description Generation (NLDG). To do this, we created two NSGen variants for comparative analysis:

- **w/o CNS**: This variant excludes the CNS process from NSGen, directly uses the natural language descriptions corresponding to all neurons in the selected layer to assemble the decision path, which subsequently guides fuzzing execution.
- **w/o NLDG**: Conversely, this variant omits NLDG, relying instead on the Jaccard similarity of decision paths post-CNS to determine the generation of fault-revealing examples.

Empirical results, as detailed in Table 11, underscore NSGen's superiority over all state-of-the-art models, as evidenced by its higher number of triggered faults (#Faults) and broader class coverage (#Classes). NSGen outperforms all two variants in #Faults, with improvements ranging from 31.42% to 935.34%, demonstrating the contribution of each of the main components in NSGen. Moreover, CNS contributes more than NLDG, reflecting its major role in reducing semantic redundancy (*c.f.* Section 3). Meanwhile, the integration of NLDG allows NSGen to capture the decision-making process of the model from a fine-grained and interpretable perspective, thereby enhancing the overall performance of NSGen.

Table 11. Ablation test for NSGen in terms of #Faults and #Classes

Variants	ImageNet/ResNet50		Flowers102/MobileNet_v2		CIFAR10/VGG16_BN		CIFAR100/ResNet50	
	#Faults/#Outputs	#Classes	#Faults/#Outputs	#Classes	#Faults/#Outputs	#Classes	#Faults/#Outputs	#Classes
w/o CNS	948/955	330	4701/5277	99	0/0	0	1000/1008	90
w/o NLDG	6925/10000	687	5920/10000	102	4923/9999	10	7345/10000	99
NSGen	**9815**/9935	**751**	**8695**/9981	102*	**8393**/9995	10*	**9653**/9999	**100**

7.3 Effectiveness of NSGen with Same Fuzzing Outputs

In RQ1, we assess the effectiveness of NSGen by examining the number of faults triggered by test inputs. These inputs are generated from the same initial batch of seeds within an identical duration or a number of iterations (*c.f.* Section 6.1). During our experiments, we observed that some criteria generated only a minimal number of test inputs. To ensure an equitable comparison, we standardize the output by requiring each coverage criterion to produce the same number of test inputs (i.e., 1,000) in this experimental setup [95]. It is important to note that certain criteria could achieve maximum coverage with just a few inputs. To address this and maintain a consistent generation process, we reset the coverage for these criteria upon reaching maximum coverage if 1,000 inputs had not yet been generated, and continue the experiment until the count was met. The remaining experimental conditions mirror those of RQ1, and we evaluate the effectiveness of each criterion using three key metrics: the number of triggered faults (#Faults), the covered classes (#Classes), and output impartiality (OI).

The experimental results are presented in Table 12. On the CIFAR10 dataset, the number of faults triggered by the test inputs from DSC is between 14.98% and 19.63% higher than those triggered by NSGen, though DSC shows limitations in terms of diversity. As noted in prior studies [90], DSC tends to favor inputs that notably diverge from the training data. In the context of fuzzing, this results in increasingly rare test inputs and a significant bias in the output distribution. Conversely, NSGen proves to be the most potent criterion on the CIFAR100, Oxford 102 Flower, and ImageNet datasets. Specifically, NSGen's test inputs triggered between 50.80% and 90.26% more faults than those generated by the state-of-the-art (SOTA) methods. Additionally, NSGen consistently excels in diversity metrics; for example, on the ImageNet dataset, NSGen's inputs covered 104 to 140 more classes than the SOTA model's inputs.

Table 12. Fuzzing results. Faults, Outputs, Classes, and OI denote triggered faults, fuzzing outputs, covered classes, and output impartiality, respectively. Best assessments are **marked**.

Models	Criteria	CIFAR10			CIFAR100			Oxford 102 Flower			ImageNet		
		#Faults/#Outputs	#Classes	OI	#Faults/#Outputs	#Classes	OI	#Faults/#Outputs	#Classes	OI	#Faults/#Outputs	#Classes	OI
ResNet	NC	564/1000	10	0.9692	723/1000	**97**	–	539/1000	102	0.9186	666/1000	339	–
	KMNC	334/1000	10	0.9667	551/1000	91	–	408/1000	102	0.9424	533/1000	281	–
	NBC	367/1000	10	0.9640	542/1000	91	–	393/1000	102	0.9464	520/1000	273	–
	SNAC	371/1000	10	**0.9748**	538/1000	88	–	417/1000	102	0.9450	530/1000	294	–
	TKNC	590/1000	10	0.9714	690/1000	96	–	563/1000	101	–	633/1000	338	–
	TKNP	0/1	1	–	0/1	1	–	0/1	1	–	0/1	1	–
	CC	662/1000	10	0.9695	798/1000	96	–	417/1000	101	–	559/1000	291	–
	LSC	754/1000	10	0.8920	777/1000	93	–	442/1000	101	–	552/1000	279	–
	DSC	**957**/1000	10	0.8962	628/1000	92	–	435/1000	102	0.9478	546/1000	292	–
	MDSC	870/1000	10	0.8914	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	CAC	330/1000	10	0.9713	557/1000	94	–	362/1000	102	**0.9629**	513/1000	287	–
	NLC	330/1000	10	0.9654	523/1000	95	–	384/1000	102	0.9574	503/1000	269	–
	NSGen	800/1000	**10**	0.8601*	**945**/1000	96*	–	**587**/1000	**102**	0.9135*	**957**/1000	**409**	–
VGG	NC	667/1000	10	**0.9731**	725/1000	93	–	625/1000	101	–	685/1000	344	–
	KMNC	363/1000	10	0.9469	571/1000	87	–	440/1000	101	–	527/1000	270	–
	NBC	367/1000	10	0.9667	606/1000	90	–	507/1000	101	–	523/1000	267	–
	SNAC	393/1000	10	0.9642	626/1000	92	–	552/1000	102	0.9243	563/1000	291	–
	TKNC	622/1000	10	0.9669	806/1000	**96**	–	512/1000	102	**0.9448**	669/1000	318	–
	TKNP	1/1	1	–	0/1	1	–	1/1	1	–	0/1	1	–
	CC	642/1000	10	0.9675	782/1000	92	–	443/1000	101	–	616/1000	301	–
	LSC	885/1000	10	0.7677	675/1000	91	–	437/1000	101	–	551/1000	273	–
	DSC	**987**/1000	10	0.8127	814/1000	83	–	582/1000	102	0.9333	595/1000	292	–
	MDSC	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	CAC	315/1000	10	0.9592	531/1000	87	–	N/A	N/A	N/A	N/A	N/A	N/A
	NLC	401/1000	10	0.9352	573/1000	87	–	439/1000	101	–	540/1000	275	
	NSGen	828/1000	**10**	0.7734*	**967**/1000	91*	–	**662**/1000	**102**	0.9079*	**964**/1000	**379**	–
MobileNet	NC	436/1000	10	0.9781	642/1000	93	–	593/1000	102	0.8972	708/1000	385	–
	KMNC	362/1000	10	0.9546	565/1000	86	–	385/1000	101	–	584/1000	318	–
	NBC	400/1000	10	0.9703	572/1000	88	–	395/1000	102	**0.9562**	616/1000	326	–
	SNAC	392/1000	10	0.9582	575/1000	88	–	391/1000	102	0.9472	613/1000	341	–
	TKNC	455/1000	10	**0.9800**	705/1000	**96**	–	566/1000	102	0.8959	750/1000	400	–
	TKNP	0/1	1	–	0/1	1	–	0/1	1	–	0/1	1	–
	CC	710/1000	10	0.9667	616/1000	90	–	371/1000	101	–	610/1000	321	–
	LSC	747/1000	10	0.8944	627/1000	85	–	382/1000	101	–	577/1000	324	–
	DSC	**990**/1000	10	0.7500	618/1000	90	–	425/1000	101	–	652/1000	309	–
	MDSC	826/1000	10	0.8546	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	CAC	397/1000	10	0.9528	555/1000	93	–	366/1000	102	0.9572	614/1000	331	–
	NLC	388/1000	10	0.9455	615/1000	88	–	377/1000	101	–	609/1000	326	–
	NSGen	861/1000	**10**	0.8323*	**962**/1000	89*	–	**620**/1000	**102**	0.8849*	**965**/1000	**431**	–

7.4 Effectiveness of Decision Path Similarity

NSGen aggregates visual concepts representing DNN prediction paths from a neuron perspective and describes them through natural language. This suggests that mutated images, which exhibit pronounced divergences in decision paths, serve as more potent instruments for unearthing discrepancies and revealing faults within DNNs, and thus we refer to [30] to assess the correlation of decision path similarity with faults in DNNs. To this end, we select the model that demonstrates the best performance on each dataset. For complex datasets like Oxford 102 Flower and ImageNet-IK, we retrieve 10 samples per category from the testing set, totaling $C \times 10$ samples. For simpler datasets such as CIFAR10 and CIFAR100, we extract 100 samples per category, totaling $C \times 100$ samples. The value of C represents the number of categories. Utilizing NSGen, we delineate decision paths for these samples and calculate their similarities, yielding (*similarity*, *success*) pairs, and compute the point-biserial correlation coefficients [41] and analyze their p-values; herein, *similarity* quantifies the similarity of decision paths between the mutated and original images, while *success* denotes the model's classification of the image as an erroneous prediction. Figure 7 visualizes the relationship between decision path similarity and model faults. Decision path similarity and defect detection show a strong correlation, especially on ImageNet-IK and Oxford 102 Flower datasets, which demonstrates that mutated images showing significant differences in decision paths are more effective in revealing DNN faults, as well as reflecting to some extent that NSGen holds the potential for generating test inputs in real scenarios.

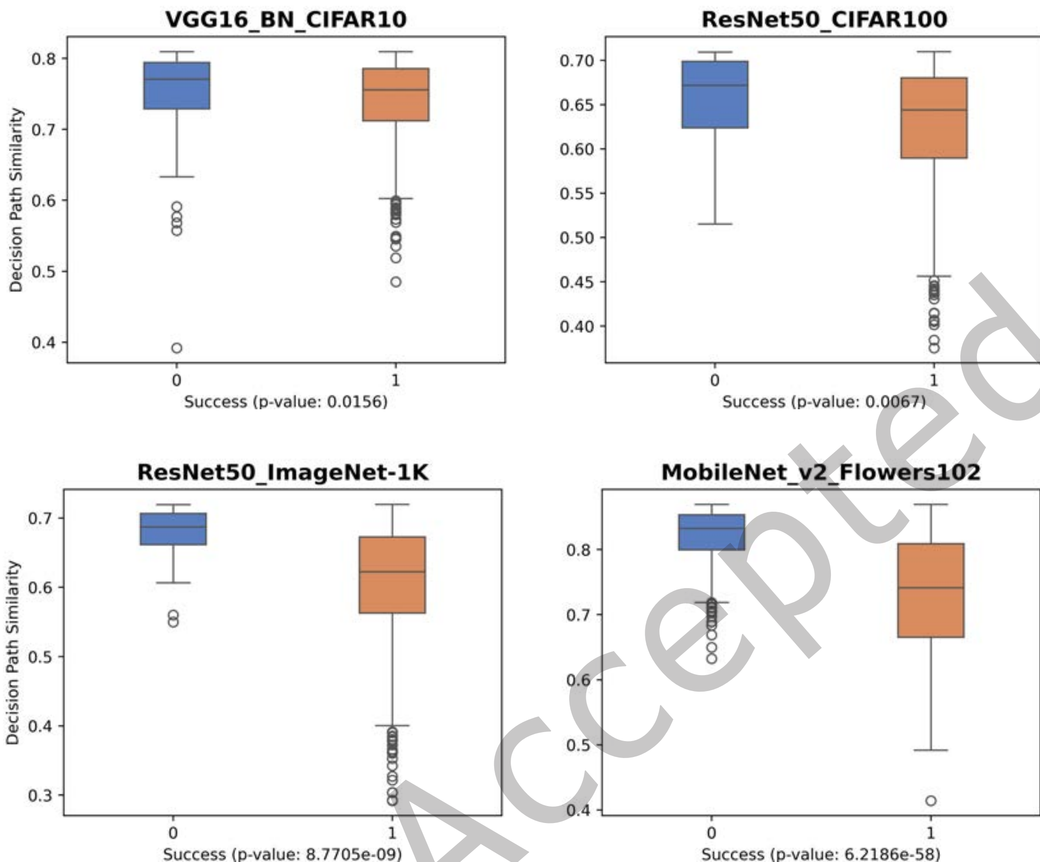


Fig. 7. Comparative Boxplot of correlation between decision path similarity and DNN bugs (x-axis shows whether the model predicted incorrectly, 1 means the prediction was wrong, and 0 is the opposite.)

7.5 Model Robustness Enhancement

We study the value of generated fault-triggering mutants, using them through retraining strategies to enhance the robustness of the target model. For each subject, we split the test set into two equal parts (S1 and S2) to avoid data leakage between the augmented training set and the evaluation set built with the same technique. Specifically, we select the first 1000 images from S1 and use NSGen to generate 1000 examples, which are all fault-revealing images. These images are integrated with the training set to form the augmented training set, which is used to retrain the original model. At the same time, we use two advanced adversarial test input generation methods, PGD [51] and BIM [43], to generate universal adversarial test inputs for S2. These methods have been widely used in existing work [37, 45, 86, 90]. Therefore, for a given subject, there are a total of three evaluation sets, namely S2, PGD→S2, and BIM→S2, which have the same size. After obtaining the retrained model, we measured its accuracy on each of the above three evaluation sets to measure its ability to reduce failures.

Table 13 demonstrates the effectiveness of mutants generated by NSGen in enhancing model robustness. The first row of the table (except column S2) identifies the evaluation data sets constructed using the corresponding techniques. Among them, the Ori row and NSGen row show the accuracy of the original model and the retrained model on each evaluation set respectively. We observe that NSGen can generally significantly enhance the model's resistance to test sets generated by PGD and BIM attack algorithms on various datasets such as CIFAR10, CIFAR100, Flowers102, and ImageNet-1K. For example, for the Flowers102 data set, the ResNet50 model improved from the initial 0% accuracy to an astonishing 43.56% on the PGD→S2 test set. Similarly, the model improved from 19.49% to 83.06% on the BIM→S2 test set. This significant improvement highlights the potential of NSGen. On average, the performance of models retrained by NSGen improved by 1.76% on the S2 evaluation set, 8.55% on PGD→S2, and 20.45% on BIM→S2. These data not only fully verify the practicality and effect of NSGen technology in improving the model's defense capabilities against various adversarial attacks, but also demonstrate its broad applicability in practical applications.

Table 13. Accuracy after adversarial training: 'Ori' denotes the original model, 'S2' denotes the equalized test set, 'PGD→S2' and 'BIM→S2' denote test sets generated by PGD and BIM attack algorithms, respectively. ↑ indicates accuracy improvements, and ↓ denotes decreases.

Model	Dataset	Criterion	S2	PGD → S2	BIM → S2
ResNet50	CIFAR10	Ori	0.9392	0.0038	0.8476
		NSGen	0.9382 ↓	0.0088 ↑	0.8766 ↑
	CIFAR100	Ori	0.7436	0.0016	0.5910
		NSGen	0.7296 ↓	0.0034 ↑	0.6080 ↑
	Flowers102	Ori	0.8277	0.0	0.1949
		NSGen	0.8806 ↑	0.4356 ↑	0.8306 ↑
	ImageNet-1K	Ori	0.7606	0.0002	0.3204
		NSGen	0.7686 ↑	0.0003 ↑	0.4002 ↑
VGG16_BN	CIFAR10	Ori	0.9392	0.0196	0.8288
		NSGen	0.9422 ↑	0.0232 ↑	0.8706 ↑
	CIFAR100	Ori	0.7298	0.0008	0.5210
		NSGen	0.7196 ↓	0.0050 ↑	0.5818 ↑
	Flowers102	Ori	0.7383	0.0	0.0904
		NSGen	0.8299 ↑	0.4072 ↑	0.7593 ↑
	ImageNet-1K	Ori	0.7338	0.0002	0.2089
		NSGen	0.7404 ↑	0.0002	0.2549 ↑
MobileNet_v2	CIFAR10	Ori	0.9374	0.0002	0.7532
		NSGen	0.9386 ↑	0.0052 ↑	0.8372 ↑
	CIFAR100	Ori	0.7394	0.0	0.4272
		NSGen	0.7354 ↓	0.0008 ↑	0.5306 ↑
	Flowers102	Ori	0.8319	0.0	0.1507
		NSGen	0.9093 ↑	0.1626 ↑	0.8109 ↑
	ImageNet-1K	Ori	0.7208	0.0002	0.2223
		NSGen	0.7210 ↑	0.0002	0.2500 ↑
Average			0.0176 ↑	0.0855 ↑	0.2045 ↑

7.6 Unveiling Faults Detected by NSGen

In this section, we delve into NSGen's distinctive capability to identify faults that other established criteria overlook. We select the ResNet50/ImageNet combination for our analysis, as NSGen has demonstrated optimal performance with this setup in prior experiments. To discern and understand the unique faults identified by NSGen, we employ UMAP [52] visualizations of the faults generated during the fuzz testing process. For comparative analysis, we choose KMNC, DSC, and NLC as benchmark metrics due to their strong performance in previous experiments. Notably, NSGen is capable of detecting error types that KMNC, DSC, and NLCC tend to ignore, as represented by the red dots Figure 8.

Further investigation into these distinct red dot clusters reveals that NSGen excels at generating test inputs that mislead the model into making incorrect predictions through spurious correlation between features and labels [54, 80]. For instance, as illustrated in the class activation maps and their accompanying descriptions (refer to Table 10), NSGen identifies semantic features such as “stick” and “eyes,” which initially lead to the classification of the subject as an “animal on a tree.” Subsequently, the feature “beak” narrows it down to “bird,” culminating in the final identification as “bulbul” based on the “head” feature. However, when the original image is altered, the model’s top layer misinterprets lower-level features like “ear” and “black-and-white stripes” as indicative of “indri,” due to the species’ characteristic black and white fur and arboreal habitat. This erroneous association causes the model to classify any animal with similar features as likely being an “indri,” showcasing how NSGen adeptly detects spurious correlations between features and labels by tapping into the neuron’s semantic decision-making process. This highlights NSGen’s unique ability to expose complex faults that other metrics might not capture, emphasizing its value in enhancing model robustness against deceptive inputs.

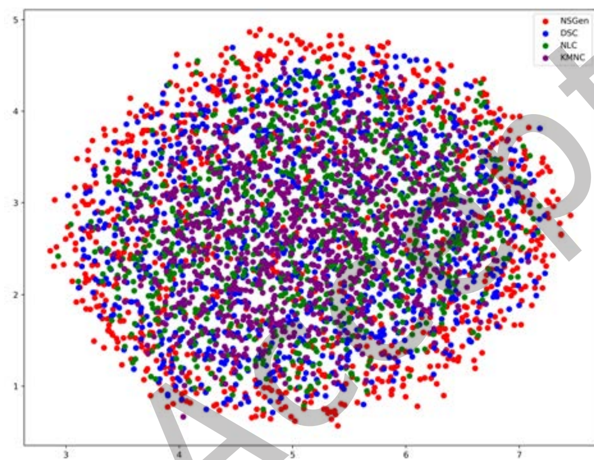


Fig. 8. The UMAP visualization of mutated features.

7.7 Accuracy Evaluation of Neuron-Description Generation

We validated the accuracy of neuron-description pairs using a dual evaluation strategy of quantitative and qualitative evaluation. Specifically, we used the ResNet50 model, known for its leading accuracy on the ImageNet dataset, as the subject of our study. For the purpose of evaluation, we randomly selected 100 neuron-description pairs generated by NSGen. The evaluation panel consisted of three Ph.D. students and six M.S. students, each with a background in computer vision and DNN-related projects. The average time for a participant to complete the evaluation was estimated to be about 25 minutes.

During the evaluation phase, participants were shown the 15 most active images associated with a neuron and asked the question: “Does the generated description: ‘{}’ accurately match this set of images?” Participants could answer “yes”, “maybe”, or “no”. These responses were then converted into numerical scores according to the following scheme: yes = 1, maybe = 0.5, and no = 0. The collective results of this experiment showed an average score of 0.8561 across all evaluations, confirming the effectiveness and precision of the neuron-description generation mechanism within NSGen. In addition, Figure 9 provides a qualitative representation of the neuron-description pairs generated by NSGen. From a qualitative perspective, these illustrative examples shed light on the model neuron’s ability to recognize and articulate salient features across different image categories. For example, the neuron-description pairing for 'ResNet-ImageNet, layer 4, neuron 570' adeptly encapsulates the quintessence of mountainous terrain, while 'ResNet-ImageNet, layer 4, neuron 1725' accurately describes the recurring motif of toilet paper in different scenarios. Similarly, the images for 'ResNet-ImageNet, layer 1, neuron 112' highlight the model's ability to identify and encapsulate the theme of blue-colored objects in different environments. The accuracy of these neuron-description pairs plays a pivotal role in guiding the fuzz testing process by pinpointing inputs that traverse distinct decision-making pathways within a DNN. This, in turn, unveils potential faults and vulnerabilities. Furthermore, the lucidity of these descriptions significantly augments the interpretability of the DNN's decision-making mechanism, offering a more intuitive grasp of model behavior.

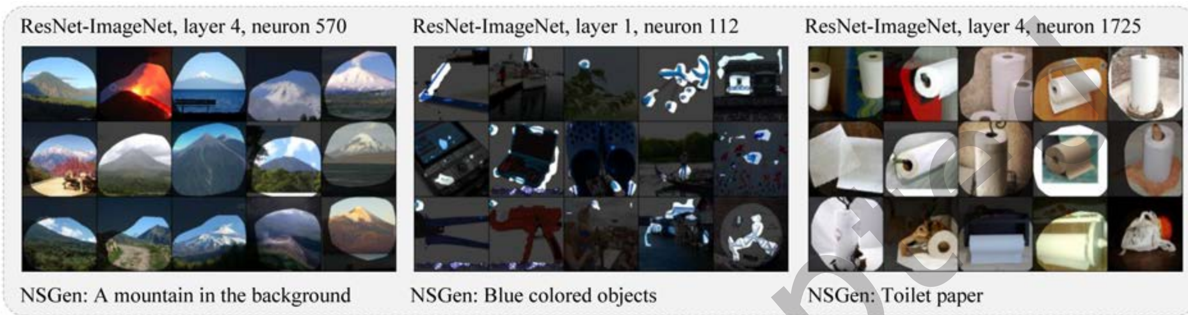


Fig. 9. Examples of the neuron-descriptions pairs.

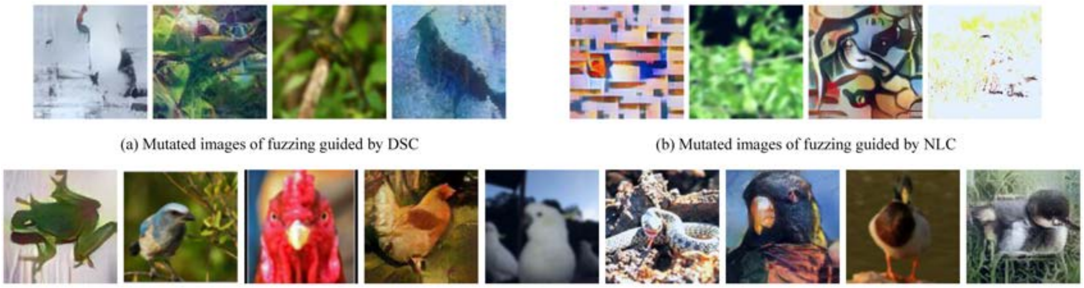
7.8 Case Study of Mutated Images' Quality

Previous research [30] has emphasized the importance of the quality of mutated images. Therefore, we randomly select some images from the first 20 mutated images. Figure 10 illustrates mutated images obtained from fuzzing guided by different criteria, including partial criteria such as DSC and NLC, as well as NSGen. DSC aims to generate test inputs that deviate from the distribution of existing training data, which could lead to the generation of images that lie outside the dataset's boundaries, making them inconsistent with the real image distribution and resulting in unnatural appearances.

On the other hand, NLC attempts to approximate the distribution of neuron outputs from a layer perspective. However, it may lose valuable neuron-specific information, leading to visually incoherent and unnatural regions in the mutation images. In contrast, NSGen utilizes the semantic information of neurons, resulting in mutation images with better-captured image features and structure. This approach reduces the risk of information loss and, as a result, mutation images guided by NSGen exhibit higher quality.

7.9 Threats to Validity

Internal Threats: Our study faces internal threats to validity related to our implementation, including aspects such as neuron selection, natural language description generation, template construction, coverage criteria, and experimental framework. Neuron selection relies on gradient attribution, which, as suggested by [13], may occasionally veer away from the data manifold, partly emphasizing background information over image characteristics. This potential threat is acknowledged, and we mitigate it by using templates (*c.f.* Section 4.3.2) to supplement the most crucial semantic information, specifically the class predicted by the DNN, thus enriching the descriptions of image features. Another internal threat concerns the accuracy of the neuron descriptions generated by NSGen (*c.f.* Section 4.2). These descriptions, derived from models trained on open-source datasets (i.e., MILANNOTATIONS [32]), might not consistently align with the specific semantic functions of the neurons



(a) Mutated images of fuzzing guided by DSC

(b) Mutated images of fuzzing guided by NLC

(c) Mutated images of fuzzing guided by NSGen

Fig. 10. Mutated images of fuzzing guided by different criteria.

in the tested DNNs. Such inconsistencies could detract from the interpretability and validity of the insights provided by NSGen. To mitigate this threat, we validate the accuracy of neuron description pairs in Section 7.7 using a dual evaluation strategy of quantitative and qualitative assessment. This review process ensures that the descriptions accurately reflect the semantic features captured by the neurons. Meanwhile, to minimize risks in our implementation, we conduct an evaluation of the correctness and adhere to the experimental framework outlined in the literature [90].

External Threats: External threats encompass potential issues regarding our choice of assessment objects and tools. Specifically, the selection of datasets and models can introduce uncertainty. To mitigate this, we choose widely used datasets (CIFAR10, CIFAR100, Oxford 102 Flower, and ImageNet) and employ three commonly utilized DNNs (VGG16_BN, ResNet50, and MobileNet_v2). ImageNet, being a large-scale dataset, is included to enhance the diversity. These choices align with prior studies on DNN test input generation [29, 82, 90]. Additionally, we acknowledge the potential threat posed by mutation rules (in RQ1, RQ2, and RQ3). To address this concern, we align with the setup used in previous research [38, 90]. The validity of generated data is another area of concern, and to mitigate this risk, we strictly adhere to the experimental settings outlined in [90]. Moreover, utilizing existing tools introduces potential threats. Given that our method, the models used, and the tools in reference [90] are all PyTorch-based, we extend the neuron coverage criteria and manually re-implement the reference [34] in PyTorch Tool. We release the experimental code for review by fellow researchers to alleviate these concerns.

Construction Threats: Construction threats primarily stem from randomness, baseline selection, and parameter choices. To address randomness, we repeat each experiment five times in each RQ and calculate the average results. We also compare NSGen with state-of-the-art methods to showcase the benefits of NSGen. Lastly, hyperparameters for NSGen and baselines could introduce validity threats. To minimize these, we align with the settings used in existing works [34, 38, 45, 82, 90] for baselines. For NSGen, we systematically adjust hyperparameters τ to suit different datasets. In the future, we plan to evaluate our approach with more configurations. The application of NSGen could be a threat. NSGen has been primarily tailored and evaluated for image classification tasks (*c.f.* Section 2.1) using convolutional neural networks (CNNs). CNN neurons are specialized for detecting local features through convolutional filters. In contrast, neurons in recurrent neural networks (RNNs) and Transformers employ recurrent or self-attention mechanisms to manage temporal and inter-positional dependencies in sequential data. This structural divergence means that the current version of NSGen may not be directly applicable to models with RNN and Transformer architectures. To mitigate this threat, we can explore the utilization of large language models to interpret neurons within smaller models effectively [11]. This approach can enhance NSGen's adaptability to a broader range of network structures.

8 RELATED WORK

The realm of DNN testing has garnered considerable attention, with various innovative approaches emerging in recent years. Notably, DeepXplore [59] introduced a pioneering white-box testing technique guided by the coverage criterion NC. Building upon this foundation, DeepGauge [45] extended NC and put forth a set of more fine-grained coverage criteria, including KMNC. Subsequently, inspired by these seminal works, a multitude of DNN testing endeavors have concentrated on the development of diverse coverage criteria. Examples include DeepCover [74], DeepCT [49], DeepMutation [50], DeepPath [78], Surprise Coverage [37, 38], Causal-Aware Coverage [34], Neural Coverage [90], and layer-level coverage criteria [69].

Building upon these criteria, a range of testing techniques have been devised to generate test cases that aim to enhance coverage, collectively referred to as Coverage-Guided Testing (CGT) techniques [25, 77]. Notably, TensorFuzz [56] employed approximate nearest neighbors algorithms for coverage calculation. DeepHunter [82] introduced novel seed sampling strategies while integrating coverage criteria from DeepGauge [45]. DLFuzz [29] emerged as the pioneer of differential fuzzing testing frameworks, focusing on input mutation to maximize neuron coverage and prediction discrepancies simultaneously. In a related vein, DeepJanus [66] characterized the frontier of DNN misbehaviors by identifying pairs of inputs that are closely related, one leading to a correct DNN output and the other to a DNN failure. The latest neuron coverage, such as Surprise Adequacy (SA) metrics [38], aim to evaluate the "surprise" level of a new input by measuring the distance between its neuron output trace and the traces of all training data. On the other hand, Neural Coverage (NLC) [90] approximates the distribution of neuron outputs from a layer perspective.

SINVAD [36] ventured into the realm of search-based input space navigation, harnessing Variational Autoencoders (VAEs) to construct a plausible input space mirroring the true training distribution. SINVAD navigates this space in pursuit of images that meet specific criteria while retaining plausibility. DeepHyperion [98] defined feature spaces tailored to DNN systems and employed Illumination Search to identify high-performing test cases via map cells representing the feature space. Additionally, reference [26] harnessed generative machine learning to create diverse test inputs that vary in high-level features, allowing the detection of failures that elude other methods. DeepHyperion-CS [99] enhanced DeepHyperion by promoting inputs that contributed more significantly to feature space exploration during previous search iterations.

It is noteworthy that numerous DNN testing works have concentrated on the design of coverage criteria and test input generation algorithms [79, 83, 94] to uncover vulnerabilities in DNN systems. However, to the best of our knowledge, there exists limited research explicitly dedicated to leveraging the semantic information of critical neurons to guide the generation of test inputs.

9 CONCLUSION AND FUTURE WORK

In this paper, we propose NSGen, an innovative neuron semantic-guided test generation approach specifically designed for DNN fuzzing. The primary objective of NSGen is to generate targeted test inputs that focus on critical neurons within the DNN by utilizing natural language descriptions. To assess the effectiveness of NSGen, we conducted a comprehensive set of experiments using three real-world DNN models. The experimental results demonstrate that NSGen exhibits a remarkable increase in the number of triggered faults by 21.4% to 61.2% compared to the state-of-the-art coverage-guided fuzzing criterion. Additionally, the test inputs generated by NSGen effectively pinpoint the faults in DNNs by specifically targeting critical neurons. These findings underscore the significant potential of neuron semantic-guided testing for DNNs, offering valuable insights for further research and development in the domains of DNN testing. In the future, we will evaluate our approach to more deep learning tasks, DNNs, and datasets. We will also strengthen our approach by exploring the utilization of a large language model or a model and dataset-independent method for generating natural language descriptions.

ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China (No. 62372071), the Chongqing Technology Innovation and Application Development Project (No. CSTB2022TIAD-STX0007 and No. CSTB2023TIAD-STX0025), the Natural Science Foundation of Chongqing (No. CSTB2023NSCQ-MSX0914) and the Fundamental Research Funds for the Central Universities (No. 2023CDJKYJH013).

REFERENCES

[1] Accessed: 2024. NSGen. https://github.com/unknownhl/NSGen.

[2] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. 2014. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing* 22, 10 (2014), 1533–1545.

[3] Uber Accident. 2018. After Fatal Uber Crash, a Self-Driving Start-Up Moves Forward.

[4] Reduan Achtibat, Maximilian Dreyer, Ilona Eisenbraun, Sebastian Bosse, Thomas Wiegand, Wojciech Samek, and Sebastian Lapuschkin. 2023. From attribution maps to human-understandable explanations through concept relevance propagation. *Nature Machine Intelligence* 5, 9 (2023), 1006–1019.

[5] Mike Aizatsky, Kostya Serebryany, Oliver Chang, Abhishek Arya, and Meredith Whittaker. 2016. Announcing OSS-Fuzz: Continuous fuzzing for open source software. *Google Testing Blog* (2016).

[6] Nadia Alshahwan and Mark Harman. 2014. Coverage and fault detection of the output-uniqueness test selection criteria. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. 181–192.

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[8] Nicholas Bai, Rahul A Iyer, Tuomas Oikarinen, and Tsui-Wei Weng. 2024. Describe-and-Dissect: Interpreting Neurons in Vision Networks with Language Models. *arXiv preprint arXiv:2403.13771* (2024).

[9] Earl T Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2014. The oracle problem in software testing: A survey. *IEEE transactions on software engineering* 41, 5 (2014), 507–525.

[10] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Agata Lapedriza, Bolei Zhou, and Antonio Torralba. 2020. Understanding the role of individual units in a deep neural network. *Proceedings of the National Academy of Sciences* 117, 48 (2020), 30071–30078.

[11] Steven Bills, Nick Cammarata, Dan Mossing, Henk Tillman, Leo Gao, Gabriel Goh, Ilya Sutskever, Jan Leike, Jeff Wu, and William Saunders. 2023. Language models can explain neurons in language models. *URL https://openaipublic. blob. core. windows. net/neuron-explainer/paper/index. html.(Date accessed: 14.05. 2023)* (2023).

[12] Marcel Böhme, Van-Thuan Pham, and Abhik Roychoudhury. 2016. Coverage-based greybox fuzzing as markov chain. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1032–1043.

[13] Sebastian Bordt, Uddeshya Upadhyay, Zeynep Akata, and Ulrike von Luxburg. 2023. The Manifold Hypothesis for Gradient-Based Explanations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3696–3701.

[14] Gerlof Bouma. 2009. Normalized (pointwise) mutual information in collocation extraction. *Proceedings of GSCL* 30 (2009), 31–40.

[15] Andrew Brock, Jeff Donahue, and Karen Simonyan. 2018. Large scale GAN training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096* (2018).

[16] Oliver Chang, Abhishek Arya, Kostya Serebryany, and Josh Armour. 2017. OSS-Fuzz: Five months later, and rewarding projects.

[17] Hongxu Chen, Yuekang Li, Bihuan Chen, Yinxing Xue, and Yang Liu. 2018. Fot: A versatile, configurable, extensible fuzzing framework. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 867–870.

[18] Edward Choi, Mohammad Taha Bahadori, Le Song, Walter F Stewart, and Jimeng Sun. 2017. GRAM: graph-based attention model for healthcare representation learning. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 787–795.

[19] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. 2012. Multi-column deep neural networks for image classification. In *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 3642–3649.

[20] Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*. 160–167.

[21] Samet Demir, Hasan Ferit Eniser, and Alper Sen. 2019. Deepsmartfuzzer: Reward guided test generation for deep learning. *arXiv preprint arXiv:1911.10621* (2019).

[22] 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.

[23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[24] Kedar Dhamdhere, Mukund Sundararajan, and Qiqi Yan. 2018. How important is a neuron? *arXiv preprint arXiv:1805.12233* (2018).
[25] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu, and Jianjun Zhao. 2019. Deepstellar: Model-based quantitative analysis of stateful deep learning systems. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 477–487.
[26] Isaac Dunn, Hadrien Pouget, Daniel Kroening, and Tom Melham. 2021. Exposing previously undetectable faults in deep neural networks. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 56–66.
[27] Xinyu Gao, Yang Feng, Yining Yin, Zixi Liu, Zhenyu Chen, and Baowen Xu. 2022. Adaptive test selection for deep neural networks. In *Proceedings of the 44th International Conference on Software Engineering*. 73–85.
[28] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 315–323.
[29] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. 2018. Dlfuzz: Differential fuzzing testing of deep learning systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 739–743.
[30] Fabrice Harel-Canada, Lingxiao Wang, Muhammad Ali Gulzar, Quanquan Gu, and Miryung Kim. 2020. Is neuron coverage a meaningful measure for testing deep neural networks?. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 851–862.
[31] 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.
[32] Evan Hernandez, Sarah Schwettmann, David Bau, Teona Bagashvili, Antonio Torralba, and Jacob Andreas. 2022. Natural language descriptions of deep visual features. In *International Conference on Learning Representations*.
[33] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[34] Zhenlan Ji, Pingchuan Ma, Yuanyuan Yuan, and Shuai Wang. 2023. CC: Causality-Aware Coverage Criterion for Deep Neural Networks. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1788–1800.
[35] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759* (2016).
[36] Sungmin Kang, Robert Feldt, and Shin Yoo. 2020. Sinvad: Search-based image space navigation for dnn image classifier test input generation. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. 521–528.
[37] 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.
[38] 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.
[39] Jinhan Kim, Jeongil Ju, Robert Feldt, and Shin Yoo. 2020. Reducing dnn labelling cost using surprise adequacy: An industrial case study for autonomous driving. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1466–1476.
[40] Igor Kononenko. 2001. Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in medicine* 23, 1 (2001), 89–109.
[41] Diana Kornbrot. 2014. Point biserial correlation. *Wiley StatsRef: Statistics Reference Online* (2014).
[42] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).
[43] 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.
[44] 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*. 165–176.
[45] Ma Lei, Juefei-Xu Felix, Sun Jiyuan, Chen Chunyang, Su Ting, Zhang Fuyuan, Xue Minhui, Li Bo, Li Li, Liu Yang, et al. 2018. Deepgauge: Comprehensive and multi-granularity testing criteria for gauging the robustness of deep learning systems. *arXiv preprint arXiv:1803.07519* (2018).
[46] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2015. A diversity-promoting objective function for neural conversation models. *arXiv preprint arXiv:1510.03055* (2015).
[47] 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.
[48] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).
[49] Lei Ma, Felix Juefei-Xu, Minhui Xue, Bo Li, Li Li, Yang Liu, and Jianjun Zhao. 2019. Deepct: Tomographic combinatorial testing for deep learning systems. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 614–618.
[50] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, et al. 2018. Deepmutation: Mutation testing of deep learning systems. In *2018 IEEE 29th international symposium on software reliability engineering (ISSRE)*. IEEE,

100–111.
[51] 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).
[52] Leland McInnes, John Healy, and James Melville. 2018. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426* (2018).
[53] Sharan Narang, Colin Raffel, Katherine Lee, Adam Roberts, Noah Fiedel, and Karishma Malkan. 2020. Wt5?! training text-to-text models to explain their predictions. *arXiv preprint arXiv:2004.14546* (2020).
[54] Yannic Neuhaus, Maximilian Augustin, Valentyn Boreiko, and Matthias Hein. 2023. Spurious Features Everywhere - Large-Scale Detection of Harmful Spurious Features in ImageNet. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 20235–20246.
[55] Maria-Elena Nilsback and Andrew Zisserman. 2008. Automated flower classification over a large number of classes. In *2008 Sixth Indian conference on computer vision, graphics & image processing*. IEEE, 722–729.
[56] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. 2019. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. In *International Conference on Machine Learning*. PMLR, 4901–4911.
[57] Tuomas Oikarinen and Tsui-Wei Weng. 2022. Clip-dissect: Automatic description of neuron representations in deep vision networks. *arXiv preprint arXiv:2204.10965* (2022).
[58] Kexin Pei. 2017. Deepxplore code release. https://github.com/peikexin9/deepxplore/.
[59] 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.
[60] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.
[61] Evelyn C Pielou. 1966. The measurement of diversity in different types of biological collections. *Journal of theoretical biology* 13 (1966), 131–144.
[62] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*. PMLR, 8748–8763.
[63] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.
[64] Yongming Rao, Wenliang Zhao, Guangyi Chen, Yansong Tang, Zheng Zhu, Guan Huang, Jie Zhou, and Jiwen Lu. 2022. Denseclip: Language-guided dense prediction with context-aware prompting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 18082–18091.
[65] Sanjay Rawat, Vivek Jain, Ashish Kumar, Lucian Cojocar, Cristiano Giuffrida, and Herbert Bos. 2017. Vuzzer: Application-aware evolutionary fuzzing.. In *NDSS*, Vol. 17. 1–14.
[66] Vincenzo Riccio and Paolo Tonella. 2020. Model-based exploration of the frontier of behaviours for deep learning system testing. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 876–888.
[67] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115 (2015), 211–252.
[68] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.
[69] Jasmine Sekhon and Cody Fleming. 2019. Towards improved testing for deep learning. In *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 85–88.
[70] Kosta Serebryany. 2016. Continuous fuzzing with libfuzzer and addresssanitizer. In *2016 IEEE Cybersecurity Development (SecDev)*. IEEE, 157–157.
[71] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[72] PR Smith. 1981. Bilinear interpolation of digital images. *Ultramicroscopy* 6, 2 (1981), 201–204.
[73] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15, 1 (2014), 1929–1958.
[74] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. 2018. Concolic testing for deep neural networks. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 109–119.
[75] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *International conference on machine learning*. PMLR, 3319–3328.
[76] Robert Swiecki and F Gröbert. 2016. Honggfuzz. *Available online a t: http://code. google. com/p/honggfuzz* (2016).

[77] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*. 303–314.
[78] Dong Wang, Ziyuan Wang, Chunrong Fang, Yanshan Chen, and Zhenyu Chen. 2019. DeepPath: Path-driven testing criteria for deep neural networks. In *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE, 119–120.
[79] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. 2018. Feature-guided black-box safety testing of deep neural networks. In *Tools and Algorithms for the Construction and Analysis of Systems: 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I 24*. Springer, 408–426.
[80] Kai Xiao, Logan Engstrom, Andrew Ilyas, and Aleksander Madry. 2020. Noise or signal: The role of image backgrounds in object recognition. *arXiv preprint arXiv:2006.09994* (2020).
[81] Xiaofei Xie, Tianlin Li, Jian Wang, Lei Ma, Qing Guo, Felix Juefei-Xu, and Yang Liu. 2022. NPC: N euron P ath C overage via Characterizing Decision Logic of Deep Neural Networks. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 3 (2022), 1–27.
[82] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. Dephunter: a coverage-guided fuzz testing framework for deep neural networks. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 146–157.
[83] Xiaofei Xie, Lei Ma, Haijun Wang, Yuekang Li, Yang Liu, and Xiaohong Li. 2019. Diffchaser: Detecting disagreements for deep neural networks. International Joint Conferences on Artificial Intelligence Organization.
[84] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*. PMLR, 2048–2057.
[85] Qin Xu, Jiahui Wang, Bo Jiang, and Bin Luo. 2023. Fine-grained visual classification via internal ensemble learning transformer. *IEEE Transactions on Multimedia* (2023).
[86] Ming Yan, Junjie Chen, Xuejie Cao, Zhuo Wu, Yuning Kang, and Zan Wang. 2023. Revisiting deep neural network test coverage from the test effectiveness perspective. *Journal of Software: Evolution and Process* (2023), e2561.
[87] Shenao Yan, Guanhong Tao, Xuwei Liu, Juan Zhai, Shiqing Ma, Lei Xu, and Xiangyu Zhang. 2020. Correlations between deep neural network model coverage criteria and model quality. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 775–787.
[88] Zhou Yang, Jieke Shi, Muhammad Hilmi Asyrofi, and David Lo. 2022. Revisiting neuron coverage metrics and quality of deep neural networks. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 408–419.
[89] Yuanyuan Yuan, Qi Pang, and Shuai Wang. 2022. Unveiling hidden dnn defects with decision-based metamorphic testing. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–13.
[90] Yuanyuan Yuan, Qi Pang, and Shuai Wang. 2023. Revisiting neuron coverage for dnn testing: A layer-wise and distribution-aware criterion. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1200–1212.
[91] Michal Zalewski. 2017. American fuzzy lop.
[92] Rowan Zellers, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. From recognition to cognition: Visual commonsense reasoning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 6720–6731.
[93] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 132–142.
[94] Peixin Zhang, Jingyi Wang, Jun Sun, Guoliang Dong, Xinyu Wang, Xingen Wang, Jin Song Dong, and Ting Dai. 2020. White-box fairness testing through adversarial sampling. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 949–960.
[95] Yuhan Zhi, Xiaofei Xie, Chao Shen, Jun Sun, Xiaoyu Zhang, and Xiaohong Guan. 2023. Seed Selection for Testing Deep Neural Networks. *ACM Transactions on Software Engineering and Methodology* 33, 1 (2023), 1–33.
[96] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. 2016. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2921–2929.
[97] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. 2017. Places: A 10 million image database for scene recognition. *IEEE transactions on pattern analysis and machine intelligence* 40, 6 (2017), 1452–1464.
[98] Tahereh Zohdinasab, Vincenzo Riccio, Alessio Gambi, and Paolo Tonella. 2021. Deephyperion: exploring the feature space of deep learning-based systems through illumination search. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 79–90.
[99] 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, 2 (2023), 1–38.