






# XMutant: XAI-based Fuzzing for Deep Learning Systems

Xingcheng Chen<sup>1,2</sup>  · Matteo Biagiola<sup>3</sup>  
 · Vincenzo Riccio<sup>4</sup>  · Marcelo  
d'Amorim<sup>5</sup>  · Andrea Stocco<sup>1,2</sup> 

Received: date / Accepted: date

**Abstract** Semantic-based test generators are widely used to produce failure-inducing inputs for Deep Learning (DL) systems. They typically generate challenging test inputs by applying random perturbations to input semantic concepts until a failure is found or a timeout is reached. However, such randomness may hinder them from efficiently achieving their goal. This paper proposes XMUTANT, a technique that leverages explainable artificial intelligence (XAI) techniques to generate challenging test inputs. XMUTANT uses the local explanation of the input to inform the fuzz testing process and effectively guide it toward failures of the DL system under test. We evaluated different configurations of XMUTANT in triggering failures for different DL systems both for model-level (sentiment analysis, digit recognition) and system-level testing (advanced driving assistance). Our studies showed that XMUTANT enables more effective and efficient test generation by focusing on the most impactful parts of the input. XMUTANT generates up to 125% more failure-inducing inputs compared to an existing baseline, up to 7× faster. We also assessed the validity of these inputs, maintaining a validation rate above 89%, according to automated and human validators.

**Keywords.** Software testing, testing deep learning systems, explainable AI

## 1 Introduction

Deep Learning (DL) systems play a crucial role in software engineering [76] due to their ability to solve complex tasks by learning from a corpus of data. However, DL systems pose additional challenges to testing. Unlike traditional software, where behavior can be anticipated through source code analysis, the actions of DL

---

<sup>1</sup> Technical University of Munich, Munich, Germany. Email: xingcheng.chen@tum.de (corresponding author), andrea.stocco@tum.de

<sup>2</sup> fortiss GmbH, Munich, Germany. Email: xchen@fortiss.org, stocco@fortiss.org

<sup>3</sup> University of St. Gallen, Università della Svizzera italiana, St. Gallen/Lugano, Switzerland. Email: matteo.biagiola@unisg.usi.ch

<sup>4</sup> Università degli Studi di Udine, Udine, Italy. Email: vincenzo.riccio@uniud.it

<sup>5</sup> North Carolina State University, Raleigh, USA. Email: mdamori@ncsu.edu

systems are less predictable [71]. Therefore, test generation for DL systems is critical for ensuring their reliability and correctness. This process involves creating a diverse set of input data to uncover potential weaknesses or biases in the model, such as incorrect predictions or failures to generalize from data samples that are different from those available during training. Testing is particularly important in applications where DL is used for decision-making in critical scenarios, such as autonomous driving [6, 27, 83, 85]. Exhaustive testing of DL systems is impractical because of the size of the input space. Thus, generating effective test cases for these systems is an important problem, which requires a comprehensive understanding of the DL model’s architecture, its training data, and the application domain.

Prior work proposed solutions to generate test cases automatically for DL systems [1, 5, 6, 12, 27, 39, 47, 53, 55, 56, 62, 72, 103]. Some approaches such as DeepXplore [67], DLFuzz [32] and DeepTest [86] target DL image classification systems and they involve raw input manipulation techniques that modify/corrupt pixel values of an input. These techniques do not generate new functional inputs as they produce minimal, often imperceptible changes to the original inputs, and are therefore suitable to test robustness and security deficiencies of the DL system [59, 73]. In contrast, functional test generation focuses on creating new inputs that deviate significantly from the original training distribution. These inputs target the long-tail problem of DL testing [97], testing the DNN’s ability to generalize to novel, unseen scenarios. Instances of functional test generators are the model-based approaches (also called semantic-based approaches) like DeepJanus [72], DeepHyperion [104] and DeepMetis [70] or latent space manipulation techniques like SINVAD [42, 43] and CIT4DNN [17]. They generate new inputs with mutations that are randomly applied to a semantic representation of the inputs [72], or to a latent vector [42]. While random mutations can eventually produce inputs that expose failures, these solutions are extraneous to the internal state of the DL system. Indeed, finding failures with black-box approaches is time-consuming, especially in computationally demanding contexts, such as simulation-based testing of self-driving cars.

This paper investigates the development of more effective and efficient semantic-based test generators for DL systems, leveraging eXplainable Artificial Intelligence (XAI) techniques. While methods similar to XAI, leveraging neuron activation values and gradient information, have been explored for raw input manipulation [57, 94, 96] to derive adversarial attacks for robustness testing, their application to semantic-based functional test generation remains unexplored. Therefore, we focus on semantic-based approaches that utilize an input model, as they have been successfully applied to various DL systems (e.g., classification, regression) and input types (e.g., text, images, logical driving scenarios).

In this work, we leverage the post-hoc local explanations of DL systems for individual predictions. Local explanations reveal the contributions of features in the input with respect to the model prediction [19]. Depending on the model’s input, the explanations can take the form of feature contributions (e.g., for textual inputs), or visual heatmaps (e.g., for image inputs) [38, 74, 75]. Previous work highlights the valuable insights provided by XAI, particularly for understanding and debugging DL systems [22, 23, 82, 87]. In our work, we leverage their information for guiding test generation by introducing XMUTANT, a DL testing technique that leverages XAI’s explanations to derive challenging inputs. More specifically, XMUTANT leverages a novel mutation operator that uses the local explanation on

a given input to identify the area of its semantic representation that has higher contributions to the decision-making process of DL systems. Our work shows that fuzzing such attention areas with targeted minor modifications accelerates fault exposure and preserves validity (i.e., the realism of the inputs) and label (i.e., the oracle). XMUTANT uses the local explanations in two ways. First, it uses them to rank and select the candidate semantic concepts from the semantic representation of input for mutation. Second, it uses these concepts to direct the mutation in the area of most attention of the DL system, enabling the approach to perform targeted input modifications.

We have evaluated XMUTANT on three DL systems, representative of diverse DL tasks, namely sentiment analysis, digit recognition, and advanced driving assistance. These case studies differ in the forms of inputs (text, images, logical scenarios) and testing levels (model-level and system-level), resulting in different semantic-based input representations and XAI explanations. In our experiments, accounting for more than  $2k$  test cases, XMUTANT shows superior effectiveness and efficiency compared to DeepJanus, a state-of-the-art semantic-based test generator that has been successfully applied to multiple domains, yet its mutation operators are not aided by any guidance. The guidance of the local explanations allows XMUTANT to expose a higher number of failure-inducing inputs (up to +208% for sentiment analysis, +125% for digit recognition and +27% for advanced driving assistance) within half the iteration budget. Moreover, on average, XMUTANT is also faster at exposing failure-inducing inputs for all DL systems (up to  $3\times$  times faster for sentiment analysis,  $7\times$  times faster for digit recognition and  $2\times$  times faster for advanced driving assistance). Our study also reveals that the failure-inducing inputs by XMUTANT are valid in-distribution inputs, according to state-of-the-art automated input validators, and that they exhibit a high validity rate ( $\approx 90\%$ ) and label preservation rate ( $\approx 70\%$ ), for human assessors. Additionally, our evaluation revealed that gradient-guided raw input manipulation methods, despite utilizing neuron activation values and gradient information, only produce corrupted inputs. In contrast, XMUTANT produces more natural failure-inducing inputs that remain within the original data manifold in semantic space.

Our paper makes the following contributions:

**Approach.** A mutation operator for semantic-based fuzzing of DL systems based on local explanations that XAI techniques produce.

**Technique.** An implementation of our approach for focused fuzzing of DL systems, implemented in the publicly available tool XMUTANT [68]. To the best of our knowledge, this is the first solution that uses XAI techniques for semantic-based focused test generation, both at the model and system level.

**Evaluation.** An empirical study on three different DL systems showing that XMUTANT is better than a state-of-the-art approach, in terms of higher effectiveness, efficiency, validity, and label-preservation rates.

## 2 Background

Deep Neural Networks (DNNs) are increasingly used in complex safety-critical tasks, such as autonomous driving [9], autonomous aviation [41], medical diagnosis [100] or disease prediction [101]. This paper leverages XAI techniques for

testing DL systems, i.e., systems that use DNNs.<sup>1</sup> In the following, we provide background on foundational concepts used in this paper: (1) semantic-based input representation and (2) XAI methods.

## 2.1 Semantic-based Input Representation

Semantic-based test generation approaches utilize *abstract aspects* of the inputs, such as the shape of a digit in an image or the sentiment polarity of a word in a text, rather than relying on *concrete aspects* of the inputs, such as changes of pixel values in an image or letter modifications in a word [70, 104, 105]. Semantic-based approaches include model-based techniques [72], which utilize domain-specific models to generate test inputs. These methods can translate the input from its concrete representation to its abstract representation and back. By manipulating inputs in this abstract space, perturbations gain semantic meaning, allowing for controlled perturbations that reflect meaningful changes in the original data. Semantic-based input representation brings an important benefit for test generation—it reduces the dimensionality of the search space. Conceptually, there exists a 1-to-many mapping from abstract to concrete values, enabling the smaller input space to be more exhaustively covered. Furthermore, semantic validity criteria are expressed at the model level, preventing automated algorithms from creating unrealistic inputs that fall outside the valid operational domain of the DL system [40, 73, 98].

## 2.2 XAI Methods

XAI methods are used to make the decisions of complex DL systems more transparent and understandable [18, 31, 90], which is essential for validation, trust-building, and regulatory compliance. In this paper, we consider XAI methods and use cases (sentiment analysis, digit recognition, and advanced driving assistance systems or ADAS) established in prior work on DL testing [70, 72, 105].

For sentiment analysis, we consider the local explanations produced by LIME [69], SmoothGrad [81], and Integrated Gradients [84]. In brief, LIME [69], namely local interpretable model-agnostic explanation, explains the predictions of DL systems by approximating their behavior with a simpler, interpretable, surrogate model in a local region. However, a large number of perturbations and inferences are required in this process, which makes LIME computationally expensive. Therefore, we also consider other gradient-based XAI methods (SmoothGrad and Integrated Gradients) to ensure the efficiency of test generation. These approaches are representative of different families of XAI methods and address known issues of saliency methods, such as gradient discontinuity and saturation. Particularly, SmoothGrad [81] reduces the noise of gradient-based explanations by adding artificial noises and averaging them, whereas Integrated Gradients [84] overcomes the gradient saturation by summing over scaled inputs.

Handwritten digit recognition and advanced driving assistance are imagery tasks, for which we consider saliency or pixel attribution methods [54, 79, 80]. These solutions generate local explanations in the input space, identifying regions of an

---

<sup>1</sup> This paper uses the terms DNNs and DL systems interchangeably, for simplicity of exposition.

Table 1: Testing characterization for three popular tasks often solved with DL systems.

	Sentiment analysis	Digit recognition	Advanced driving assistance
Testing level	Model	Model	System
DNN task	Classification	Classification	Regression
Test input	Text	Digit image	Road in a driving simulator
Input to DL system	Word tokens	Digit image	Driving scenario
Semantic representation	Words	Control points (image)	Control points (road center)
Failure criterion	Misclassification	Misclassification	Out of road bounds

input image that influence the decision-making process of DNNs. Particularly, in addition to using SmoothGrad [81] and Integrated Gradients [84] mentioned before, we also considered CAM (Class Activation Map)-based methods, as the models under test are typically convolutional neural networks. For example, Grad-CAM++ [11] generates heatmaps by computing a weighted combination of the positive partial derivatives from the last convolutional layer.

### 3 XMutant

XMUTANT is a focused fuzzing technique for DL systems. It leverages post-hoc local explanations for individual predictions to guide fuzzing. XMUTANT is applicable to any DL system where semantic representations are available. As mentioned in Section 2.1, we instantiate XMUTANT for tasks established in prior work on DL testing [70, 72, 105] for which semantic models exist: (1) sentiment analysis, (2) digit recognition, and (3) advanced driving assistance systems (ADAS).

**Testing scenarios.** Table 1 shows the characteristics of our testing scenarios. Semantic representations are often domain-specific. For example, both sentiment analysis and digit recognition are classification tasks, but the semantic modelings are different. In sentiment analysis, the words in a movie review are mapped to an embedding space whereas in digit recognition, the bitmap of a MNIST handwritten digit [15] is converted into sequences of cubic Bézier curves [24] defined by a series of control points, according to the Scalable Vector Graphics representation. For advanced driving assistance, the centerline of a road in a driving scenario is represented by Catmull-Rom cubic splines [10], also specified by a sequence of control points.

The different testing levels also imply distinct interpretations of failures. For sentiment analysis and digit recognition, a failure is defined as the disagreement between the output of the DNN under test and the ground truth, e.g., a misclassification of the movie review or the handwritten digit. For advanced driving assistance, failures are identified by the misbehavior of the whole DL system, i.e., the autonomous vehicle, in response to the outputs of the DNN, rather than isolated incorrect predictions on individual images. Following existing research [6], we evaluate whether a sequence of incorrect DNN predictions leads to a violation of the safety requirements of the system, i.e., the vehicle driving out of the road, or a reduction of the driving quality [37].

Notably, unlike evolutionary fuzzing approaches, XMUTANT does not employ a continuous fitness score to rank or retain test inputs. Instead, it solely relies on a binary failure criterion to decide whether to keep a mutated input. The intention is to isolate and evaluate the impact of XAI-guided mutation alone, without the

**Algorithm 1: XMUTANT: XAI-guided Fuzzing**

**Input:** DNN under test, set of test inputs  $T$ , failure criterion, termination condition.  
**Output:** Failure set  $F$

```

1 Initialize test inputs  $T$  with random individuals if  $T = \emptyset$ ;
2 Initialize failure set  $F = \emptyset$ ;
3 while  $T \neq \emptyset$  or termination condition do
4   for each test input  $t$  in  $T$  do
5     Test DL system with the input  $t$ ;
6     if failure criterion not met then
7        $S \leftarrow \text{GETSEMANTICREPRESENTATION}(t)$  ;
8        $e \leftarrow \text{LOCALEXPLANATIONCOMPUTATION}(t)$  ;
9        $cs \leftarrow \text{SEMANTICCONCEPTSELECTION}(S, e)$  ;
10       $d \leftarrow \text{MUTATIONDIRECTIONCOMPUTATION}(S, e, cs)$  ;
11       $S \leftarrow \text{MUTATE}(S, cs, d)$  ;
12       $t \leftarrow \text{GENERATECONCRETEINPUTFROMSEMANTIC}(S)$  ;
13    else
14      Move  $t$  from  $T$  to  $F$ ;
15    end
16  end
17 end
18 return  $F$ ;

```

confounding effects of fitness-based selection. This design also preserves flexibility for integrating XMUTANT into larger testing frameworks that may employ their own prioritization or fitness metrics.

**Algorithm.** Algorithm 1 shows the main steps of XMUTANT. XMUTANT takes as input the DL system, a set of test inputs  $T$ , the failure criterion, and a termination condition, such as a time budget or a number of generated inputs. XMUTANT produces as output a set  $F$  containing failure-inducing inputs on the DNN under test. The test inputs in  $T$  are used to test the DL system. Initially, XMUTANT initializes the set of test inputs  $T$ , e.g., by randomly selecting seeds from a test dataset, when they are available (for sentiment analysis or digit recognition), or by randomly generating inputs if they are unavailable (for the advanced driving assistance system). If an original test input  $t$  produces a failure, it is discarded. The main loop of the algorithm (Lines 7–12) retrieves the semantic representation  $S$  (Line 7), which is a sequence of semantic concepts, and the local explanations  $e$  (Line 8), which will be used for focused mutations. Then, a candidate semantic concept  $cs \in S$  is selected for mutation (Line 9), based on the area of highest attention indicated by the local explanation  $e$ , followed by a mutation of the semantic concept in such area (Lines 10–11). Finally, the newly generated semantic-based representation is restored to the original input space (Line 12) and used for testing the DL system. In case of failure, the algorithm stores the evolved test input  $t$  to the failure set (Lines 14). Finally, the algorithm returns the failure set  $F$ .

### 3.1 XAI-guided Mutation Operator

XMUTANT’s insight is to fuzz test input on the critical areas of the DNN attention to create challenging inputs to the corresponding DL system and, consequently, trigger misbehaviors. XMUTANT’s mutation operator selects the semantic concept that is near a high-attention region and, subsequently, performs a targeted mutation in such region.

XMUTANT’s mutation operator comprises three steps: (1) computing the local explanation (Line 8), (2) selecting the semantic concept to mutate (Line 9), and (3) determining the mutation direction and applying mutation (Line 10, 11). First, XMUTANT uses an XAI technique to compute local explanations associated with a DNN and its test instances (procedure `LOCALEXPLANATIONCOMPUTATION` on Line 8). Typically, the explanation of the DNN prediction for a given input has the same size as the input, with each dimension of the input space corresponding to an explanation score. Second, XMUTANT selects the semantic concept to mutate on the high-attention area of the local explanation (procedure `SEMANTICCONCEPTSELECTION` on Line 9), where an intermediate weight vector  $w$  is obtained and denotes the importance of semantic concepts with  $\dim(w) = \dim(S)$ . For textual inputs, the local explanation corresponds directly to the semantic concept, hence selecting the semantic representation is based on the explanation’s magnitude (i.e.,  $w = e$  and  $\dim(e) = \dim(w)$ ). However, for model- or system-level imagery inputs, our approach needs to map the high-dimensional explanations, onto the lower-dimensional semantic concept space through a function  $f$  before the selection, i.e.,  $w = f(e)$  and  $\dim(e) > \dim(w)$ . To ensure diversity of generated test inputs, XMUTANT randomly selects candidate semantic concepts  $cs$  based on the weights  $w$ , where semantic concepts with higher weights have higher chances of being chosen. Third, when a candidate semantic concept is selected, XMUTANT leverages the local explanations to choose the mutation direction, aiming to maximize the mutation effectiveness (Line 10).

XMUTANT applies the same XAI-guided fuzzing workflow across different DL applications, and it is configured according to the kind of input and semantic representation associated with the task under test (Table 1). We elaborate on the domain-specific configurations in terms of semantic concept selection and mutation direction in the following sections.

### 3.2 Model-level Application on Textual Inputs

For textual inputs, the inputs to the DNN are tokens corresponding to the words. Therefore, obtained explanations have the same dimensions as the semantic representation, i.e., words (`GETSEMANTICREPRESENTATION` in Algorithm 1).

With model-agnostic XAI technique methods such as LIME [69], the feature attributions for each token can be computed directly. With gradient-based XAI methods, however, the presence of a non-differentiable embedding layer in the DL model under test, makes the gradient computation unavailable. To address this, we propose a customized approach for interpreting such DL models. Let us assume that the embedding layer mapping the input vector  $x$  consisting of  $L$  tokens in the embedding space of dimension  $N$ , is a matrix of size  $\mathbb{R}^{L \times N}$ . Then we take the following steps. First, we remove the embedding layer from the model under test to obtain the corresponding submodel. Second, we apply gradient-based XAI methods to compute the explanation  $\hat{e} \in \mathbb{R}^{L \times N}$  in the embedding space for the submodel, with each row vector  $\hat{e}[i] \in \mathbb{R}^N$  representing the explanation vector for the corresponding token  $x[i]$ . Third, we map the explanation in the embedding space back to the input space with summation (for Integrated Gradients) or norm operators (for SmoothGrad), e.g.,  $e[i] = \text{sum}(\hat{e}[i])$  for each token  $x[i]$ . This process



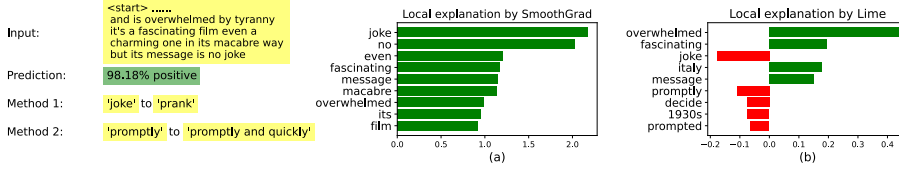


Fig. 1: Example of movie review sentiment analysis case study (best viewed in color). (a) Local explanation by SmoothGrad; (b) Local explanation by LIME.

yields the local explanation  $e$  with the same dimension as the input  $x$ , which XMUTANT later uses to guide mutations (LOCALEXPLANATIONCOMPUTATION).

The explanations are used to derive a weight vector  $w$  over tokens, from which XMUTANT samples one candidate concept  $cs$  (SEMANTICCONCEPTSELECTION). However, the local explanations obtained for textual inputs differ in their characteristics depending on the XAI method used. For instance, SmoothGrad captures the gradient information, indicating the sensitivity of the predictions to word variations (Figure 1 (a)). In this case, the weight vector consists of absolute gradient information. In contrast, LIME and Integrated Gradients provide explanations on feature attributions, which represent the magnitude of the positive (or negative) contribution of a word on the prediction, making them useful for guiding the mutation direction (Figure 1 (b)). In this case, the weight vector contains the raw explanations.

Regarding the actual mutation operation, we used two mutation methods available in the literature [105]: (1) replacing a word with its synonym obtained from WordNet [60], and (2) adding an “and” conjunction after an adjective (or adverb), followed by a synonym of the adjective (or adverb). These operators are used to perturb the selected token  $cs$ , resulting in a modified semantic input  $S$  (MUTATE). These mutation methods ensure that the original meaning of the sentence is preserved, thereby maintaining a high validity of generated test inputs.

During the fuzzing process, one of the mutation methods is selected at each iteration. When the first mutation method is chosen, we apply it to the selected word  $cs$ , regardless of the considered XAI method. This is because it is uncertain whether replacing a synonym will increase or decrease the contribution of the selected word. In contrast, the second mutation method enhances the semantics of the sentence by adding a synonym. Depending on the explanation, XMutant decides whether to apply directional mutation (e.g., amplifying or attenuating polarity) or random variation (MUTATIONDIRECTIONCOMPUTATION). Therefore, for XAI methods that differentiate feature attributions (i.e., LIME [69] and Integrated Gradients [84]), we perform targeted mutation by applying the second mutation method only to the word with opposite effects to the prediction (e.g., we modify “promptly” to “promptly and quickly” but we do not modify “overwhelmed” in Figure 1), which helps to more effectively challenge the model under test. For XAI methods that are not focused on explaining the feature attributions (i.e., SmoothGrad), we apply the second mutation method on the selected word, regardless of its effect on the prediction.

After mutation, the modified semantic input is projected back into the concrete input space—i.e., transformed into a full sentence—which serves as a new test case for the DL system (GENERATECONCRETEINPUTFROMSEMANTIC).



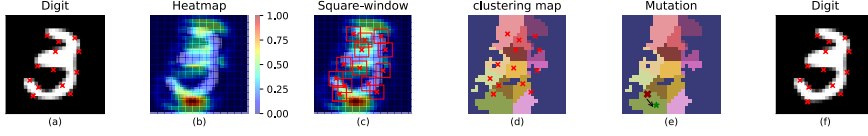


Fig. 2: XMUTANT steps applied to the MNIST case study (best viewed in color). (a) SVG semantic-based representation of a digit 3. Control points (i.e., semantic representation) are shown as red crosses; (b) Heatmap generated by Grad-CAM++; (c) Square windows centered on the control points; (d) Clustering on the local explanation (one cluster per control point); (e) Control point selection (red cross marker) and mutation towards the attention attractor of the cluster (green star); (f) Digit 3 resulting by mutating the candidate control point.

### 3.3 Model-level Application on Imagery Inputs

For imagery inputs, the semantic representation  $S$  is extracted by converting the bitmap image into a set of SVG control points (cf. [Algorithm 1: GETSEMANTICREPRESENTATION](#)). The local explanation, so-called heatmap, is itself an image with the same dimensions as the original image, where pixels are highlighted based on the attention that the DNN under test gives to each portion of the image (cf. [Algorithm 1: LOCALEXPLANATIONCOMPUTATION](#)). The subsequent selection of the candidate semantic concept to be mutated and the computation of the mutation direction are based on a single heatmap  $\mathbf{H}$ .

[Figure 2](#) shows the different steps of the XMUTANT operator for a handwritten MNIST [\[15\]](#) digit image. Given an image, XMUTANT converts the bitmap representation into an SVG representation to extract the semantic representation, i.e., a sequence of control points ([Figure 2 \(a\)](#)). Then, it computes the heatmap on this SVG representation using an XAI method (e.g., Grad-CAM++ in [Figure 2 \(b\)](#)); in the heatmap relevant locations correspond to hot color intensities (e.g., red/yellow), whereas irrelevant locations correspond to cold color intensities (e.g., blue). The heatmap values are normalized such that low-attention values are close to zero, while high-attention values are close to one.

Selecting the candidate semantic concept  $cs$ , i.e., the control point  $cp_k$ , for mutation requires the computation of the weights corresponding to the control points based on the heatmap ([Algorithm 1: SEMANTICCONCEPTSELECTION](#)). We evaluate two techniques in this paper, based on *square windows* and *clustering*.

#### 3.3.1 Square Windows Semantic Concept Selection

Square windows involve computing the weight of a control point by averaging the intensity values within a square window centered on the control point  $(x_k, y_k)$  (e.g., the red squares in [Figure 2 \(c\)](#)), according to the following equation:

$$w_k = \frac{1}{(2d+1)^2} \sum_{i=-d}^{i=d} \sum_{j=-d}^{j=d} \mathbf{H}[(\mathbf{p}_{c_k} + (i, j))], \quad (1)$$

where  $d$  is the distance between the center of the square  $(x_k, y_k)$  and each side (and the window size  $ws$  is given by  $ws = 2d + 1$ ), while  $\mathbf{p}_{c_k} \in \mathbb{R}^2$  denotes the coordinates of the control point in the heatmap. Subsequently, the weights  $w$  for each semantic concept are normalized.

The square windows approach, although straightforward, involves choosing an appropriate window size  $ws$ . When the window size is large, it can exceed the input’s boundaries or have squares of different control points overlap. If the window size is small, large regions of the local explanation might be uncovered, potentially missing some high-attention areas. Additionally, this method does not provide any guidance for the direction of mutation, thus the direction is determined randomly (**Algorithm 1**: `MUTATIONDIRECTIONCOMPUTATION`).

### 3.3.2 Clustering-based Semantic Concept Selection

To reduce reliance on hyperparameters and offer better spatial coherence, we also propose a control point selection technique based on clustering. XMUTANT uses the k-means clustering algorithm [93] to cluster pixels based on their values with the number of clusters  $n$  to match the number of control points. We modified the typical k-means process by initializing the centroids with the control points and limited the algorithm to a single iteration, without convergence. **Figure 2** (d) shows 13 clusters identified by different colors, one for each control point. Then, XMUTANT computes the weight  $w_k$  for each control point  $cp_k$  as the sum of the ratios between each pixel value in the cluster of  $cp_k$ , and the distance between the coordinates of the pixel and the respective control point:

$$w_k = \sum_{i=1}^{m_k} \frac{\mathbf{H}[\mathbf{p}_i^k]}{d_i^k}, \text{ with } d_i^k = \max(1, \|\mathbf{p}_i^k - \mathbf{p}_{cp_k}\|), \quad (2)$$

where  $m_k$  is the number of pixels in the cluster of control point  $cp_k$ ,  $\mathbf{p}_i^k \in \mathbb{R}^2$  denotes the coordinates of  $i$ -th pixel in the cluster, and  $d_i^k$  is the Euclidean distance between  $\mathbf{p}_i^k$  and the coordinates of the control point in the heatmap (i.e.,  $\mathbf{p}_{cp_k}$ ). To avoid division by zero when  $\mathbf{p}_i^k = \mathbf{p}_{cp_k}$ , we set the lower bound of  $d_i^k$  to one.

Once the weights have been normalized and the control point  $cp_k$  sampled (e.g., the control pointing the green cluster in the bottom-left corner of **Figure 2** (e)), XMUTANT mutates the selected control point towards the attention attractor  $\mathbf{c}_k$  (**Algorithm 1**: `MUTATIONDIRECTIONCOMPUTATION`), which is defined as the center of intensity of the respective cluster, given by:

$$\mathbf{c}_k = \frac{\sum_{i=1}^{m_k} \mathbf{H}[\mathbf{p}_i^k] \mathbf{p}_i^k}{\sum_{i=1}^{m_k} \mathbf{H}[\mathbf{p}_i^k]}. \quad (3)$$

Finally, the new input is obtained by mutating  $cs$  toward the attractor  $\mathbf{c}_k$  (**Algorithm 1**: `MUTATE, GENERATECONCRETEINPUTFROMSEMANTIC`).

**Figure 2** (e) shows an example of how the selected control point in the green cluster (bottom left) is mutated towards the respective attention attractor. This control point is marked by a red cross and moves toward the green star, which represents the attention attractor of the green cluster. The mutation direction is depicted by a black arrow extending from the control point to the attention attractor, leading to a contraction of the high-attention area. **Figure 2** (f) shows the resulting digit after the mutation, where the control point in the bottom-left corner of the image (see **Figure 2** (a)) has shifted further downward.

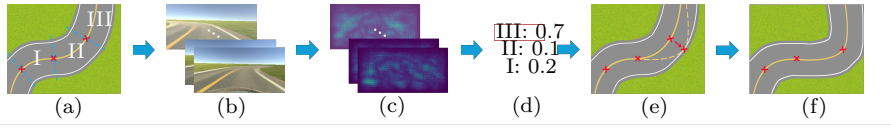


Fig. 3: XMUTANT steps for the ADAS (best viewed in color). (a) A road with 3 control points, or semantic representation (red crosses). The road sectors are indicated as Roman numerals and separated by dashed cyan lines; (b) A sequence of driving frames recorded by the vehicle’s camera; (c) A sequence of heatmaps, corresponding to the driving frames, generated by SmoothGrad; (d) Weights for each road sector and associated semantic concept; (e) Control point selection (top-right) and mutation towards the right lane (dashed yellow line indicates the new centerline); (f) Road resulting from the mutation of the control point in sector III.

### 3.4 System-level Application on Logical ADAS Scenarios

As the ADAS processes a stream of images from a driving simulator, applying our approach at an individual image level (as described previously for model-level testing) does not effectively assist in selecting and mutating candidate semantic concepts, i.e., control points. Instead, XMUTANT generates a series of heatmaps for each image perceived by the DNN, and aggregates them to retrieve a semantic score that reflects the system’s overall performance. This allows the selection of critical semantic components for mutation, following the same fuzzing loop described in [Algorithm 1](#) (cf. GETSEMANTICREPRESENTATION, LOCALEXPLANATIONCOMPUTATION, and SEMANTICCONCEPTSELECTION).

[Figure 3](#) shows the steps of XMUTANT for the ADAS. The test input is a sequence of control points located at the center of the road ([Figure 3](#) (a)), determining the shape of the two-lane road that will be instantiated in the driving simulator (we assume the other environment variables, such as weather conditions, as fixed). Once the road is instantiated, the DNN that controls the vehicle takes as input the frames that are recorded by the onboard camera ([Figure 3](#) (b)) and outputs the driving commands (i.e., steering angles). Therefore, XMUTANT computes the heatmaps of such camera frames, resulting in a sequence of heatmaps ([Figure 3](#) (c)).

To weigh the control points, we divide the road into segments extending from each control point to the road’s centerline (e.g., in [Figure 3](#) (a) there are three road sections). Each road segment is initially linked to the control point at its start. This approach is based on preliminary findings indicating that mutations at a control point predominantly affect the driving behavior in the section immediately following it. To aggregate sequences of heatmaps, XMUTANT computes the derivative of a heatmap  $\nabla \mathbf{H}_t$  at a certain time step  $t$  as the average absolute difference between the pixels of two consecutive heatmaps  $\mathbf{H}_{t-1}$  and  $\mathbf{H}_t$ :

$$\nabla \mathbf{H}_t = \frac{1}{wh} \sum_{i=1}^w \sum_{j=1}^h |\mathbf{H}_{t-1}[i, j] - \mathbf{H}_t[i, j]| \quad (4)$$

Subsequently, the sampling weight  $w_k$  of the control point  $cp_k$  is given by the mean value of all derivatives in the corresponding road section, i.e.,  $w_k = 1/m_k \sum_{i=2}^{m_k} \nabla \mathbf{H}_i^k$ , where  $m_k$  is the number of frames in a specific road section,

and  $\{\mathbf{H}_1^k, \dots, \mathbf{H}_{m_k}^k\}$  denotes the heatmaps in the road section associated to the control point  $cp_k$ .

Indeed, existing work [82] has shown that the derivative of a local explanation is indicative of a poor-performing DNN. In this way, XMUTANT grants a higher weight to road sections—and associated control points—where the driving behavior of the DNN is more erratic (cf. **Algorithm 1**: SEMANTICCONCEPTSELECTION). Consequently, by mutating such control points in the corresponding road sections, XMUTANT is more likely to generate a test input that induces more challenging driving conditions. For instance, in **Figure 3** (d), the road section with the highest mean derivative is road section III; in this example, the corresponding control point in the top-right corner of **Figure 3** (e) is sampled.

Next, XMUTANT determines the mutation direction by considering the perpendicular line to the road’s centerline at the candidate control point. Consequently, only two directions are possible: moving the control point toward the right lane or the left lane (i.e., **Algorithm 1**: MUTATIONDIRECTIONCOMPUTATION). To determine the appropriate mutation direction, we analyze the series of heatmaps for the chosen road segment. XMUTANT partitions each local explanation into two equally-sized sub-heatmaps, i.e.,  $\mathbf{H}[:w/2, :]$  and  $\mathbf{H}[w/2+1 :, :]$  (where the column separates start and end indices for each dimension of the local explanation). XMUTANT then calculates the difference in average intensity between the left and right sub-heatmaps in the road segment. The mutation direction is chosen based on the side with the highest average intensity (in **Figure 3** (e) the selected direction is the right direction). The selected control point is then shifted laterally toward that direction (**Algorithm 1**: MUTATE), resulting in a new and more challenging road configuration (**Algorithm 1**: GENERATECONCRETEINPUTFROMSEMANTIC), as visualized in **Figure 3** (f).

## 4 Empirical Study

### 4.1 Research Questions

We consider the following research questions:

**RQ<sub>1</sub> (effectiveness):** How effective is XMUTANT at finding misbehaviors?

**RQ<sub>2</sub> (efficiency):** How efficient is XMUTANT at finding misbehaviors?

**RQ<sub>3</sub> (configuration):** How do effectiveness and efficiency vary when considering different XAI algorithms? What is the best configuration of semantic concept selection and mutation direction?

**RQ<sub>4</sub> (validity):** To what extent are the inputs generated by XMUTANT valid and label-preserving?

**RQ<sub>5</sub> (comparison):** How does XMUTANT compare with gradient-guided raw input manipulation techniques?

The first research question (RQ<sub>1</sub>) assesses whether XMUTANT attains a high failure rate. The second research question (RQ<sub>2</sub>) evaluates the efficiency of XMUTANT. The third research question (RQ<sub>3</sub>) evaluates different configurations of XMUTANT obtained by varying the XAI algorithm, semantic concept selection, and mutation direction strategy. The fourth research question (RQ<sub>4</sub>) studies the usefulness of the inputs that XMUTANT produces, both for automated input validators and human assessors. The final research question (RQ<sub>5</sub>) examines how

XMUTANT compares to widely used gradient-guided raw input manipulation techniques. Although these techniques differ in their primary objectives, generalization and robustness testing are often confused since they share the same goal, i.e., DNN reliability. Thus, we analyze and compare the generated inputs from our approach and those generated by existing methods to provide evidence of the benefits of a semantic-based representation, as test inputs generated by these distinct methodologies may form different manifolds in the semantic space.

## 4.2 Objects of Study

### 4.2.1 Datasets and Models

**IMDB.** Concerning sentiment analysis, we consider a DL system designed to classify the sentiment (i.e., positive or negative) of movie reviews from the IMDB database [58]. We use the DL system available in the replication package of existing work [44], characterized by an embedding layer, an LSTM layer [33], and two fully connected layers. The model achieves 85.25% accuracy on the test set.

**MNIST.** Concerning digit recognition, the DL system classifies handwritten digits from the MNIST dataset [52]. This DL system takes 28x28 greyscale images as input and predicts the corresponding digit (the possible classes range from 0 to 9). In this paper, we test the convolutional DNN architecture provided in the replication package of existing work [72]. Architecturally, it is characterized by 2 convolutional layer, a max-pooling layer, and two fully connected layers. The model achieves 98.99% accuracy on the test set.

**ADAS.** Concerning advanced driving assistance, the DL system controls a vehicle in the Udacity simulator [88], a cross-platform driving simulator developed with Unity3D [89], widely used in the ADS testing literature [35, 37, 71, 83]. The DL system includes a DAVE-2 model (a DNN regressor), a Level 2 [36] ADAS that performs the lane keeping functionality from a training set of images collected when the driver is an expert human pilot, by predicting the corresponding driving commands imitating the human driving behavior. We obtained the trained DAVE-2 model and the simulator from the replication package of existing work [7]. The model architecture includes three convolutional layers for feature extraction, followed by five fully connected layers. The simulator supports the creation of open-loop road tracks for testing ADAS models, including the ability to generate and modify road topologies.

### 4.2.2 Comparison Baselines

To assess the guidance provided by our XAI-based approach, we compare XMUTANT against DeepJanus, a popular semantic-based test generator [72]. For the selection of semantic concepts and mutation directions, DeepJanus relies on a random method that selects a candidate semantic concept for mutation based on a uniformly weighted probability. The mutation direction is task-specific. For IMDB, it applies one of the mutation methods on a randomly selected semantic concept. For MNIST, the mutation direction is sampled from the uniform distribution over 0 to  $2\pi$ . For ADAS, the mutation direction is randomly selected from one of the two directions perpendicular to the road’s curvature. To ensure a fair comparison,

we applied the same extent value for mutation direction to both XMUTANT and DeepJanus’s mutation methods. In our study, we sampled the extent of MNIST from a uniform distribution over the interval  $[0, 1.2]$ , and for ADAS we adopted a fixed value 4, proportional to the road’s width 8. These values are determined based on existing literature [7, 72].

Regarding raw input manipulation approaches, we selected popular approaches that use gradients to guide the generation of adversarial inputs, i.e., FGSM [30], DeepXplore [67], and DLFuzz [32]. For FGSM, we progressively increased the perturbation intensity up to the 20% threshold, as higher intensities affect the validity of the generated inputs (i.e., the original images were no longer recognizable due to excessive noise or corruption). For DLFuzz and DeepXplore, we adopted the default settings described in the original papers.

#### 4.2.3 Metrics used for Analysis

Concerning RQ<sub>1</sub>, we evaluate the effectiveness of the test generation technique by computing the *cumulative failure rate* observed under a given number of mutation iterations across different configurations. The cumulative failure rate is obtained by dividing the number of failures by the total number of inputs. For IMDB and MNIST, a failure is characterized by the number of misclassifications. For ADAS, we measure the number of safety-critical failures in terms of *out of bounds* (OOBs), which occur when the vehicle drives outside the road’s drivable lanes during execution in the simulator.

Concerning RQ<sub>2</sub>, we evaluate the performance of XMUTANT by measuring the *relative efficiency* with respect to DeepJanus. Particularly, relative efficiency is computed by integrating the *cumulative failure rates* over the number of iterations to obtain the area under the cumulative failure curve (AUFC). For each configuration and number of iterations, we calculate the AUFC (Area Under the Failure Curve) and then divide it by the AUFC of our comparison baseline. Considering computational time complexity, XMUTANT introduces an additional overhead for computing local explanations compared to the baseline, while all other components in the fuzzing loop introduce the same computational expense. To assess its impact, we specifically evaluated the proportion of the XAI overhead relative to the baseline’s average computation time per iteration. To measure computational cost, we compute the *XAI overhead*, defined as the average time per iteration spent on local explanation, normalized by the baseline (DeepJanus) iteration time:

$$\text{XAI overhead} = \frac{T_{\text{XMUTANT}} - T_{\text{baseline}}}{T_{\text{baseline}}}$$

To jointly assess these two aspects, relative efficiency and computational cost, we also report a composite efficiency metric, which adjusts the relative AUFC by factoring in the per-iteration overhead, i.e.,

$$\text{composite efficiency} = \frac{\text{relative efficiency}}{1 + \text{XAI overhead}}$$

offering a more operational view on real-world efficiency under time constraints.

Concerning RQ<sub>3</sub>, we assess the effectiveness and efficiency of each configuration of XMUTANT using the metrics used for RQ<sub>1</sub> and RQ<sub>2</sub>. In RQ<sub>4</sub>, we present the

validity rates for both automated input validation and human evaluation, along with the label-preservation rates as judged by human assessors.

In RQ<sub>5</sub>, we evaluate efficiency and effectiveness of competing approaches, by calculating the number of misclassified inputs divided by the total number of original seeds, and the total elapsed time divided by the number of misclassified inputs. We also adopt two metrics to evaluate the realism of generated inputs. The density and coverage metrics [63] measure how well the generated inputs align with real data distribution, using k-nearest neighbors to define local manifolds. Particularly, coverage quantifies the proportion of real points with nearby generated points, while density reflects the concentration of generated points around the original test inputs. Following the reference implementation in Dola et al. [17] we set  $k = 5$ .

#### 4.2.4 Configurations

We evaluate a total of 24 configurations, designed to systematically assess the impact of the core components (XAI algorithm, the semantic concept selection method, and the mutation direction) of XMUTANT for the purpose of ablation-style analysis.

For IMDB, XMUTANT uses three XAI algorithms (SmoothGrad [81], LIME [69], and Integrated Gradients [84]) and two mutation methods from the literature, namely synonym replacement and addition [104].

In the MNIST classification task, XMUTANT uses three XAI algorithms (SmoothGrad [81], Grad-CAM++ [11], and Integrated Gradients [84]) and two strategies (square windows and clustering). For square windows, we use a window size value of  $ws = 3$ , which was found appropriate during preliminary experiments (e.g., such value does not lead to the creation of square windows that exceed the input boundaries), as presented in Appendix 10.1. For mutation direction, XMUTANT relies on random selection due to the absence of attractor information. The clustering strategy allows for the mutation direction to be chosen either towards the attention attractor or randomly, thereby isolating the effect of mutation direction.. Additionally, following previous works [26, 77], XMUTANT pre-processes the heatmap to discard low-intensity values and filter out the noise using a threshold  $\epsilon = 0.1$ , improving stability of semantic selection.

For the ADAS regression task, XMUTANT uses three XAI algorithms (SmoothGrad [81], Grad-CAM++ [11], and Integrated Gradients [84]). Following existing literature, it adopts the heatmap derivative function method from ThirdEye [82] to aggregate consecutive heatmaps into a single score. Considering the mutation direction, XMUTANT uses three strategies (the attractor mutation direction does not apply to ADAS). The first strategy determines mutation directions by identifying the lane receiving the most focus (High). For instance, if the heatmaps indicate higher attention on the right lane, the mutation direction chosen is to the right. In contrast, the second strategy involves mutation directions in lanes receiving the least focus (Low). For example, if the heatmaps reveal the right lane as receiving the most focus, the mutation direction is set to the left. The final strategy adopts a random approach to select the mutation direction (i.e., either left or right).

These controlled variations across tasks and configurations enable us to analyze the contribution of each component in guiding semantic-based fuzzing.



#### 4.2.5 Procedure

Concerning RQ<sub>1</sub>, RQ<sub>2</sub>, and RQ<sub>3</sub>, the evaluation procedure is as follows.

For IMDB, we randomly select 200 movie reviews from the original IMDB test set. We discard 26 reviews that are misclassified before mutation. For MNIST, we randomly select 2,000 digits from the original MNIST test set, uniformly distributed for each class. We run the classifier to ensure that all original seeds are correctly classified. We discard 23 digits identified during this screening task and evaluate the remaining seeds. For ADAS, we randomly generate 60 test inputs (i.e., roads) with the following characteristics: maximum curvature 70 degrees and 12 control points to avoid too many invalid roads and ensure the variety of choice during mutation [7]. As a sanity check, we ensured that the trained DAVE-2 model drives all roads in the simulator successfully. As a result, we discard 5 roads that trigger misbehaviors before any mutations. We execute each configuration of XMUTANT and DeepJanus, using a budget of 100 iterations for IMDB, 1,000 for MNIST, and 30 for the ADAS. This upper bound value was found through experimentation to be adequate for convergence of either XMUTANT or DeepJanus.

Concerning RQ<sub>4</sub>, we assess the validity of XMUTANT’s output using automated validators and questionnaires for human assessment. Particularly, for IMDB, we perform the validity check using ChatGPT [66], a state-of-the-art large language model. For textual inputs, manual validation is inherently subjective and can lack consistency. Additionally, the workload for manual evaluation is significant, potentially leading to decreased evaluation quality. Large language models like ChatGPT, on the other hand, provide a viable automated solution for validation and are increasingly regarded as reliable as humans for comprehending and explaining small chunks of text, such as those produced by XMutant for IMDB, as highlighted by recent studies [91, 102].

We used ChatGPT-4o-mini [65] since it is cost-efficient and shows strong performance on textual intelligence and reasoning tasks. We append the textual input (to be classified) to the following prompt “*Assume you are a sentiment classifier, given a text of a movie review removing Stopwords and Punctuations. Please only reply ‘positive’, ‘negative’, or ‘invalid’ if the sentence does not make sense.*”. Due to the inherent randomness of ChatGPT, we repeated the validity assessment five times for each input and reported the average score.

For MNIST, we randomly select 200 misclassified inputs produced by the best configuration of XMUTANT and the DeepJanus baseline. We assess the validity of XMUTANT’s and DeepJanus’s output using SelfOracle [83], a distribution-aware input validator for imagery data, which has shown high agreement with human validity assessment in a large comparative study about test input generators for DL [73]. We obtained the trained model of SelfOracle for MNIST from the replication package of the paper by Riccio and Tonella [73]. We applied SelfOracle to reconstruct all failure-inducing images by each variant of XMUTANT and DeepJanus using the same rate of false alarms as the original study ( $\epsilon = 0.05\%$ ). We also evaluate validity and label preservation with human assessors. We conducted a questionnaire where 10 participants were asked to identify a digit, with choices ranging from 0 to 9, or to indicate if it was unrecognizable as a digit. This method allowed us to assess both the validity (if the response is a digit) and label preservation (if the response matches the intended label).

For ADAS, we did not perform any automated or manual validation of the roads, as our test generation process filters out invalid roads by ensuring they meet specific criteria: (1) the start and end points are different; (2) the road is contained within a square bounding box of a predefined size (specifically  $250 \times 250$ ), and (3) there are no intersections.

Concerning RQ<sub>5</sub>, we conduct the comparison only for the image classification task, as the comparing techniques do not apply to textual inputs and logical driving scenarios. We use the same original 2,000 seeds for all techniques, for which we compute the effectiveness and efficiency metrics. For coverage analysis, we randomly selected 50 generated failure-inducing inputs per class, totaling 500, to ensure that coverage-related metrics are not affected by various population sizes. However, for DeepXplore, we only obtain 44 generated inputs out of 2,000 original seeds, so we analyze all available samples.

Since our analysis prioritizes semantic similarity instead of pixel-wise similarity, we conduct coverage analysis in an embedding space that captures high-level semantic features. We use a pretrained VGG16 model on ImageNet as an embedding extractor, as it has demonstrated 100% accuracy in MNIST classification through transfer learning [49], confirming its ability to capture relevant digits features. The 512-dimensional embedding vectors are then visualized by reducing their dimensionality using principal component analysis (PCA), to show the overlap between generated and real data distributions.

## 4.3 Results

### 4.3.1 Effectiveness, efficiency, configuration (RQ<sub>1</sub>, RQ<sub>2</sub> and RQ<sub>3</sub>)

Concerning effectiveness (RQ<sub>1</sub>), Table 2 presents the effectiveness result (cumulative failure rate) for all configurations of XMUTANT and DeepJanus as the baseline, for all case studies (IMDB, MNIST, and ADAS), over different iterations. For all case studies, all configurations of XMUTANT outperform the baseline, regardless of the iteration considered. We assessed the statistical significance of the differences in the cumulative failure rate between XMUTANT and the baseline using the non-parametric Mann-Whitney U test [92] (with  $\alpha = 0.01$ ) and the magnitude of the differences using the Cohen’s  $d$  effect size [14]. The differences were found to be statistically significant ( $p$ -value  $< 0.01$ ), with varying effect sizes.

**RQ<sub>1</sub>:** *The guidance provided by the XAI allows XMUTANT to generate significantly more failure-inducing inputs than DeepJanus (up to +208% for sentiment analysis, up to +125% for digit recognition, and +27% for system-level advanced driving assistance).*

Concerning performance (RQ<sub>2</sub>), Table 3 presents (1) the relative efficiency, i.e., the ability to induce failures quickly, measured by the relative AUFC w.r.t. DeepJanus; (2) the XAI overhead, i.e., the additional per-iteration cost introduced by XAI computations as a ratio of the baseline’s average computation time per iteration; and the joint metric composite efficiency, which reflects the real-world time effectiveness by balancing AUFC gains and computational costs. Regardless of the XAI overhead, all configurations of XMUTANT were producing misbehaviors faster than the baseline, as evidenced by the relative efficiencies regarding

Table 2: Results for RQ<sub>1</sub> Effectiveness. The value indicates the *Cumulative Failure Rate [%]* over different Iterations. P-values indicating statistically significant differences are highlighted in **bold**.

	Iterations	SmoothGrad		LIME			Integrated Gradients			DJ	
IMDB	20	21.84		32.18			31.03			9.77	
	40	39.08		50.57			56.90			17.82	
	60	46.55		62.64			66.67			24.14	
	80	50.57		69.54			73.56			28.74	
	100	54.60		76.44			75.86			32.18	
	p-value	6.93E-18		1.17E-17			7.90E-18			-	
	effect size	large		large			large			-	
		SmoothGrad		Grad-CAM++			Integrated Gradients			DJ	
		Clustering		SW			Clustering		SW	-	
	Iterations	Attractor	Random	Random	Attractor	Random	Random	Attractor	Random	Random	
MNIST	200	77	24	24	74	21	22	60	27	26	
	400	94	47	45	94	46	45	80	51	47	
	600	97	62	60	98	60	61	87	66	61	
	800	98	71	69	99	70	73	90	76	71	
	1000	99	78	76	99	77	80	92	82	77	
	p-value	4.94E-165	1.09E-164	3.46E-165	2.57E-164	1.19E-161	2.10E-164	4.86E-165	7.03E-165	7.05E-165	
	effect size	large	small	small	large	small	small	large	medium	small	
		SmoothGrad			Grad-CAM++			Integrated Gradients			DJ
	Iterations	Random	Low	High	Random	Low	High	Random	Low	High	
ADAS	10	65	60	67	75	87	75	64	65	58	
	20	89	82	84	95	93	95	87	85	75	
	30	91	89	95	95	95	95	95	87	89	
	40	95	89	95	95	95	95	95	93	95	
	p-value	8.23E-08	5.26E-06	1.05E-07	1.07E-07	1.10E-07	1.09E-07	1.05E-07	1.54E-07	1.07E-07	
	effect size	small	small	small	medium	medium	medium	small	small	small	

DeepJanus as the relative efficiencies are greater than 1. The results hold for all case studies.

Considering the XAI overhead, LIME is extremely time-consuming since it requires estimating a surrogate model, making it unsuitable for test generation. 17 out of 21 configurations of XMUTANT with other XAI methods remains more efficient than DeepJanus, even when the overhead is accounted for, reported by the composite efficiencies larger than 1. Besides, in system-level ADAS testing, XAI computations constitute a relatively small portion of the total execution time, as executing the driving simulation is computationally expensive. These results show XMUTANT’s time-efficiency, especially for complex system-level testing settings.

**RQ<sub>2</sub>:** XMUTANT is faster than DeepJanus at exposing failure-inducing inputs for DL systems (ranging from 2× to 7× times faster for model-level sentiment analysis and digit recognition, and 2× faster for system-level advanced driving assistance).

Concerning the configurations of XMUTANT (RQ<sub>3</sub>), in the case of IMDB, when XMUTANT is equipped with XAI methods that explain the feature attribution (Integrated Gradients and LIME), it outperforms the baseline both in terms of cumulative failure rate and relative efficiency. However, since the computation time by LIME has shown to be computationally expensive, the configuration using Integrated Gradients is more practical for real-world scenarios.

For MNIST, the performance of XMUTANT shows negligible differences in both cumulative failure rate and relative efficiency, regardless of the XAI method used, when relying on different control point selection strategies such as Clustering or

Table 3: Results for RQ<sub>2</sub> Efficiency. The value indicates the *Relative Efficiency* (AUFC Ratio) over different Iterations, followed by XAI Computation Overhead and composite time-to-failure efficiency compared to the DeepJanus. The Iteration used for the computation of composite efficiency is underlined. The composite efficiency values outperforming baseline are highlighted in **bold**.

	Iterations	SmoothGrad			LIME			Integrated Gradients			DJ
IMDB	10			1.93			1.44			2.05	1
	20			2.08			2.31			2.71	1
	40			<u>2.12</u>			<u>2.82</u>			3.05	1
	100			1.90			2.74			2.59	1
	XAI overhead ↓			0.35			236.63			1.47	0
	Composite Efficiency ↑			<b>1.57</b>			0.01			<b>1.23</b>	1
	Iterations	SmoothGrad			Grad-CAM++			Integrated Gradients			DJ
		Clustering		SW	Clustering		SW	Clustering		SW	-
		Attractor	Random	Random	Attractor	Random	Random	Attractor	Random	Random	-
MNIST	100	7.00	1.83	2.11	<u>4.81</u>	1.03	1.32	5.54	2.06	<u>2.15</u>	1
	200	6.37	1.77	1.82	5.54	1.32	1.48	4.92	2.00	1.97	1
	400	4.03	1.52	1.47	3.84	1.34	1.36	3.26	1.67	1.57	1
	1000	2.12	1.23	1.20	2.09	1.19	1.21	1.86	1.32	1.23	1
	XAI overhead ↓	0.87	0.87	0.87	0.79	0.79	0.79	0.52	0.52	0.52	0
	Composite Efficiency ↑	<b>3.74</b>	0.98	<b>1.13</b>	<b>2.69</b>	0.58	0.74	<b>3.64</b>	<b>1.36</b>	<b>1.41</b>	1
	Iterations	SmoothGrad			Grad-CAM++			Integrated Gradients			DJ
		Random	Low	High	Random	Low	High	Random	Low	High	-
ADAS	5	<u>2.00</u>	<u>2.20</u>	2.87	<u>2.80</u>	<u>2.53</u>	2.07	<u>2.13</u>	<u>2.53</u>	<u>2.60</u>	1
	10	1.35	1.36	1.69	1.86	1.78	1.60	1.57	1.60	1.56	1
	20	1.31	1.21	1.36	1.47	1.53	1.48	1.30	1.30	1.32	1
	40	1.15	1.09	1.18	1.23	1.25	1.23	1.16	1.12	1.14	1
	XAI overhead ↓	0.19	0.19	0.19	0.16	0.16	0.16	0.16	0.16	0.16	0
	Composite Efficiency ↑	<b>1.68</b>	<b>1.85</b>	<b>2.41</b>	<b>2.41</b>	<b>2.18</b>	<b>1.78</b>	<b>1.83</b>	<b>2.18</b>	<b>2.24</b>	1

Square Windows. These configurations consistently achieved a cumulative failure rate over 75% after 1,000 iterations. Notably, when equipped with clustering-based control point selection and random mutation direction, Integrated Gradients proved to be the best configuration. However, the introduction of a mutation direction towards the attractors of the heatmaps significantly enhances XMUTANT effectiveness. All XAI methods, aside from Integrated Gradients, reached a failure rate of 70% in less than 200 iterations. This performance is on par with that achieved by DeepJanus over 1,000 iterations.

Concerning the ADAS task, the relative efficiency of various configurations with XMUTANT is approximately twice that of the baseline. However, combining with heatmap-guided mutation direction did not significantly enhance XMUTANT. Among four different XAI methods, GradCAM++ performed best, achieving a 95% cumulative failure rate in only 15 iterations.

**RQ<sub>3</sub>:** *For model-level digit recognition, the best configurations of XMUTANT are SmoothGrad, and Grad-CAM++ equipped with clustering selection and mutation towards the Attractor. For system-level advanced driving assistance, the best configurations of XMUTANT are Grad-CAM++ with the Random mutation or towards the High attention lane.*

#### 4.3.2 Validity (RQ<sub>4</sub>)

Figure 4 reports the validity rate for IMDB and MNIST. Figure 4 (a) presents the validity assessment by ChatGPT, including the mean and variance due to

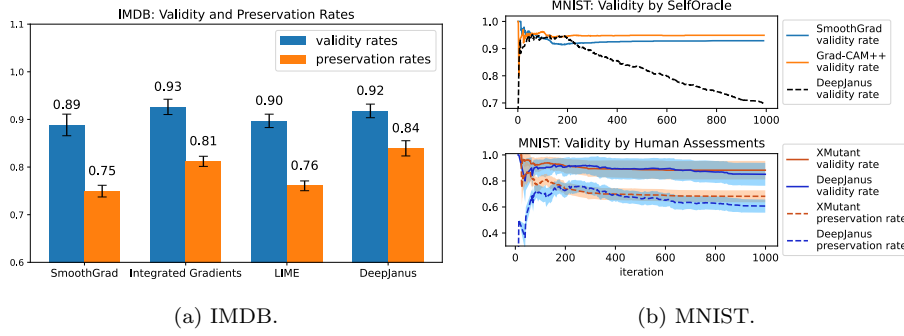


Fig. 4: Validity and preservation rates for sentiment analysis (left) and digit recognition (right).

ChatGPT’s inherent randomness. All configurations of XMUTANT and DeepJanus maintain a high validity (above 89%) and preservation rate (above 75%) with relatively low deviations since the adopted mutation methods are considered conservative and can preserve the input semantics. DeepJanus has a slightly higher preservation rate as it was assessed on a smaller set of inputs (Table 4.3.1).

For the automated validator of MNIST, we only report the best configurations found in the previous research questions, namely SmoothGrad Clustering Attractor and Grad-CAM++ Clustering Attractor. The plot (Figure 4 (b) top) shows that the validity rate for the DeepJanus steeply decreases with the increasing number of iterations. This is expected because DeepJanus produces out-of-distribution inputs if random mutations are applied an excessive number of times (particularly after 200 iterations). Differently, all configurations of XMUTANT produce failure-inducing inputs that are regarded as in-distribution for SelfOracle. It is important to notice that the validity rate for XMUTANT stabilizes at 95% (the used validity threshold in SelfOracle), which indicates that the distribution of the misclassified inputs generated by XMUTANT highly overlaps with the distribution of the original testing dataset of MNIST. This is attributed to the XMUTANT’s ability to generate failure-inducing inputs by modifying the original digits slightly, within minimal targeted modifications due to the XAI guidance.

Figure 4 (b) (bottom) reports, at varying iteration steps, the validity and label preservation rates by the human assessors. The plot shows the average value and the variance between assessors. The results from the human assessment differ from those of the automated detector. XMUTANT still outperforms DeepJanus, with its validity and label-preserving rate converging around 0.9 and 0.7, respectively, after 300 iterations. In comparison, DeepJanus’s validity and label preservation rate are lower than that of XMUTANT and exhibit a downward trend.

**RQ4:** *The failure-inducing inputs by XMUTANT are regarded as valid in-distribution inputs, according to state-of-the-art automated input validators. Moreover, they exhibit a high validity rate ( $\approx 90\%$ ) and label preservation rate ( $\approx 70\%$ ), according to human assessors.*

Table 4: Comparison of XMutant and gradient-guided adversarial testing methods.

Technique	Effectiveness	Efficiency (s)	Density	Coverage
DLFuzz	60.10%	0.65	0.2156	0.34
FGSM	43.80%	0.37	0.0104	0.05
DeepXplore	0.02%	76.84	0.2591	0.11
XMUTANT	<b>74.15%</b>	<b>0.33</b>	<b>0.3824</b>	<b>0.41</b>

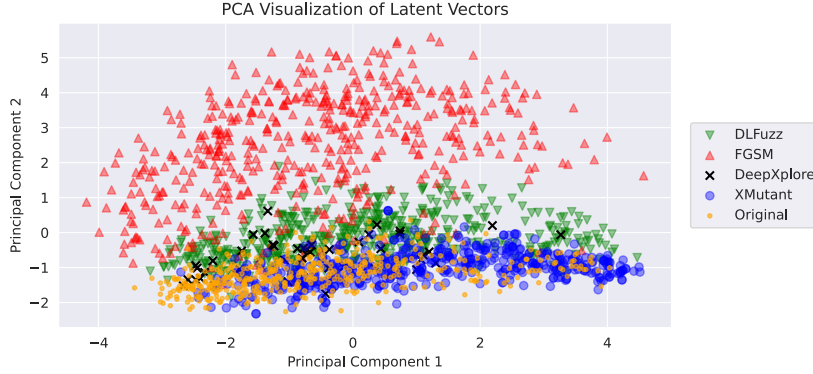


Fig. 5: Semantic space visualization of generated inputs by different techniques.

#### 4.3.3 RQ5 (Comparison)

Table 4 shows, for the various techniques, the results for effectiveness, efficiency, density, and coverage. In this analysis, we studied the best performing configuration of XMUTANT (Grad-CAM++ in combination with Clustering and Attractor). XMUTANT outperforms all other gradient-guided methods also in terms of efficiency and effectiveness. DeepXplore generates a low amount of failures since it only inserts noise in the upper-left occlusion window of the image.

Figure 5 shows that adversarial techniques exhibit a significant shift in the second principal component (PC2) in the PCA space, while the first principal component (PC1) remains relatively aligned with the original data. The failure-inducing inputs generated by XMUTANT generally adhere to the original distribution and fill in sparsely populated regions of the original distribution (towards the right part). PC1 captures global features such as overall brightness and coarse-grained structure, whereas PC2 is more sensitive to local features such as texture details. Despite minor pixel modifications, RIM techniques seem to generate radically different images in the PCA space. It can be observed that the density and coverage are generally consistent with the PCA visualization, indicating that the inputs generated by the XMUTANT are closest to the original distribution.

Figure 6 compares the input produced by XMUTANT and the other three raw input manipulation methods (DLFuzz, FGSM, and DeepXplore). These pixel-level fuzzing approaches inject different patterns of noise, either in the global background or locally focused on some specific areas, while largely preserving the semantic content of the input (the shape of the digit) from the original seed. In contrast, XMUTANT operates on a semantic input representation, modifying the

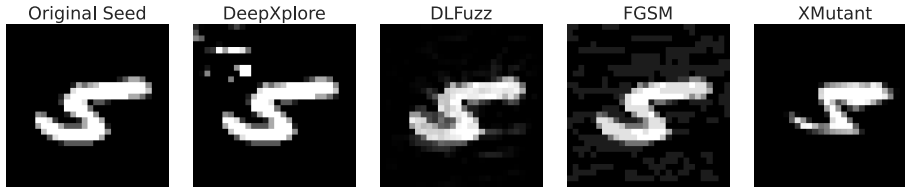


Fig. 6: An example of generated failure-inducing inputs by raw input manipulation techniques and XMUTANT

shape of the digit through control-point mutations. This leads to functionally new and visibly distinct test inputs challenging the model under test.

**RQ5:** *XMUTANT outperformed 3 gradient-guided adversarial testing methods on four metrics. The failure-inducing inputs by XMUTANT are able to better cover the original distribution in semantic space.*

#### 4.4 Threats to Validity

##### 4.4.1 Internal validity

We conducted comparisons between all variants of XMUTANT and the baseline within the same experimental framework, using identical benchmarks. A potential internal validity is our implementation of the scripts used to assess these results, which have undergone extensive testing. Additionally, regarding the ADAS model and the simulation platform, we utilized artifacts available from the replication packages of prior studies [7].

##### 4.4.2 External validity

The limited number of DL systems in our evaluation poses a threat in terms of the generalizability of our results. To mitigate this, we considered DL systems addressing different tasks, input data, and testing levels. For the local explanation methods, we considered a limited number of XAI methods. To address this threat, we selected XAI algorithms from diverse families (e.g., surrogate, gradient, and saliency-based methods). Our results show that XMUTANT consistently outperforms the baseline regardless of the XAI method used. However, it is important to note that the optimal choice of the XAI algorithm is domain-specific.

## 5 Qualitative Analysis

We analyzed some of the failures qualitatively to understand the reasons why the local explanations are able to guide the mutation process effectively. We focused our analysis on the digit recognition and advanced driving assistance systems.





(a) Qualitative analysis for MNIST by SmoothGrad



(b) Qualitative analysis for MNIST Grad-CAM++

## 5.1 Qualitative Analysis of Digit Recognition

For model-level digit recognition, we investigate the underlying relationship between model confidence and heatmaps by observing how they change with different mutation configurations. Since heatmaps are high-dimensional matrices, we em-

ploy two metrics to extract relevant information. The first metric is the average intensity, for which we have to remove the normalization applied during local explanation computation to restore the original data. The second metric is Shannon’s entropy [78], which captures the level of disorder in the heatmap. A lower entropy value suggests more concentrated or structured intensity patterns, whereas a higher entropy value indicates a more uniform distribution of intensity values. The first metric reflects the raw intensity comparison across a series of heatmaps, while the second metric quantifies internal variations within a single heatmap.

We analyzed two XAI methods, SmoothGrad (Figure 7a) and Grad-CAM++ (Figure 7b), selecting three mutation configurations guided by the same seed. The first observation is related to **gradient attenuation**, as the mean intensity of both methods decreases with the confidence, especially those driven by XMUTANT. Notably, Grad-CAM++’s mean intensity drops significantly, thus it is visualized in a logarithmic scale. Since SmoothGrad reflects gradients across the entire model, the gradient attenuation is less pronounced w.r.t. Grad-CAM++, which instead focuses on local gradients at the last convolutional layer, directly exposing the gradient attenuation in this layer and resulting in a more pronounced decline. Our second observation is related to **entropy degradation**. The inputs generated by XMUTANT cause the entropy to increase when the confidence decreases, indicating that the DNN progressively loses its focus. This phenomenon is also more observable with the mutation guided by XMUTANT equipped with SmoothGrad, since it generates noise-averaged gradients, whereas the entropy of heatmap generated by Grad-CAM++ appears noisy.

## 5.2 Qualitative Analysis of ADAS

For system-level advanced driving assistance, we considered three configurations of XMUTANT—Grad-CAM++ (Random), Grad-CAM++ (High)—and further analyzed the driving quality degradation before/after mutation.

Additionally, to allow a fine-grained comparison across approaches for ADAS, we assess the driving quality degradation (DQD) of the lane-keeping model using two metrics [37]: (1) the maximum CTE value (cross-track error, i.e., the lateral position of the vehicle w.r.t. the center road) and (2) the Euclidean norm of the steering angle output in the segment of the road affected by the mutation. We record the driving data on the road segment relevant to the candidate control point and derive the driving quality value by applying the selected norm to the raw driving data. Consequently, we calculate DQD by subtracting the pre-mutation driving quality from the post-mutation driving quality.

For each technique, we considered 6,622 pairs of pre- and post-mutation driving quality quantities, involving the maximum CTE value and the Euclidean distance of steering angle in the mutated road segments.

Figure 8 depicts the distribution of DQD, i.e., the subtraction of driving quality pairs. DeepJanus produces only minor variations, with smaller means and a tighter distribution, with its median closely aligned with symmetry around zero. On the other hand, the mutations led by XMUTANT result in larger mean and variance values for DQD, as well as more data points distributed in areas with high values, indicating a higher driving quality degradation due to more challenging road topologies. Within each approach, we conduct the above-mentioned statistical test

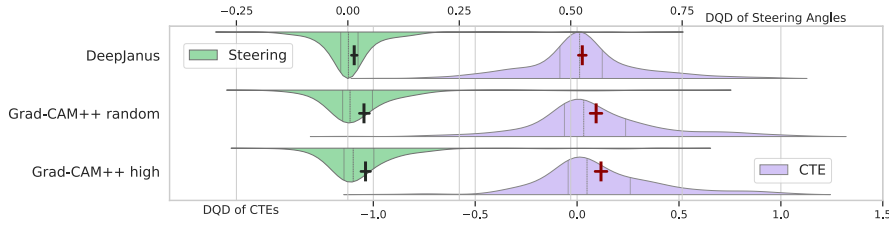


Fig. 8: Distributions of Driving Quality Degradation (DQD) based on CTEs/steering angles.

over the driving quality distributions before and after mutation. The differences in the quality of driving before/after mutation were found to be statistically significant according to the non-parametric Mann-Whitney U test [92] ( $p$ -value  $< 0.01$ ), which is not the case for the DeepJanus approach.

## 6 Discussion

**XAI for DL Test Generation.** In common practice, XAI’s explanations are used qualitatively by humans to understand how a DNN processes the inputs. However, in our research, we use them quantitatively, under the assumption that they contain relevant information related to the behavior of DNNs [75, 87]. Our experiments showed the effectiveness brought by XAI in guiding semantic-based fuzzing, as focusing on the area of attention led to faster discovery of misbehaviors while retaining their validity, as the mutations are more focused and less disruptive.

Although different XAI methods may vary significantly in their quality and behavior, particularly in terms of established evaluation metrics such as faithfulness and sensitivity [2, 54], our results demonstrate that XMUTANT consistently improves failure discovery across most configurations. This suggests that XMUTANT does not require the explanations to be fully accurate or faithful, but rather benefits from the heuristic signal provided by these explanations to increase the likelihood of failure-inducing inputs. This is due to the nature of fuzzing itself, which is inherently exploratory and tolerant of imperfect guidance. Therefore, XMUTANT remains robust even when the underlying XAI techniques vary in their quality and behavior.

Additionally, understanding the characteristics of these explanations by different XAI methods is crucial for XMUTANT to leverage them for effective mutation selection and direction. For example, attribution-based methods are particularly useful for determining mutation direction, while sensitivity-based methods are helpful to guide selection toward more influential input components. XMUTANT adapts its mutation strategy accordingly to accommodate these differences in explanation semantics.

The results of this study showed the significant benefits of utilizing XAI for semantic-based fuzzing. XAI enables the identification of DNN’s misbehaviors while ensuring that the tests remain valid and relevant to the original input domain, thanks to the semantic representation and focused test generation. Although the generation of local explanation introduces additional computational overhead, our findings suggest that the trade-off is justified. Our effectiveness results, even

for complex applications such as advanced driving assistance systems, indicate that the benefits of XAI in failure exposure outweigh the costs of computing heatmaps. Our experiments suggest that incorporating XAI (except for the model-agnostic XAI method LIME) into the testing workflow is beneficial. Additionally, the adopted open-source XAI methods can be further optimized for efficiency by parallelizing multiple backpropagations or DNN inferences. We will evaluate this aspect in our future work.

**XMutant’s Configurations.** All configurations of XMUTANT are stable in terms of effectiveness across all iterations but our study shows that the choice of the optimal XAI algorithm is task-specific.

For model-level sentiment analysis, both LIME and Integrated Gradients can generate a significant amount of failures, as their explanations differentiate between the positive and negative impacts of semantic concepts, allowing for applying targeted mutations. However, due to LIME’s high computational cost, Integrated Gradients is the most suitable candidate for XMUTANT in practice for textual inputs. For model-level digit classification, all XAI algorithms are effective, with SmoothGrad (Clustering Attractor) showing the best results. We believe this is associated with its more direct method of reflecting gradient information within the input space, which significantly influences the DNN’s outcome during model-level testing. For system-level automated driving assistance, our findings indicate that Integrated Gradients did not provide benefits compared to the baseline. Conversely, CAM-based algorithms proved highly effective and efficient if a low testing budget was available. However, in system-level testing, it is challenging to identify a reasonable direction for mutation, due to cumulative uncertainty and flakiness of the system [3]. This results in targeted mutations (High versus Random) being less effective and offering limited guidance since the system may exhibit different behaviors across consecutive test runs.

**XAI-guided Tests preserve validity and original distribution semantics.** Thanks to the usage of semantic-based input representation (i.e., a model), XMUTANT generates failure-inducing inputs while retaining the essential properties of validity and label accuracy. As a result, the generated inputs show a more natural distribution in the embedding space while exhibiting *functional novelty*, serving as an effective mean to evaluate the model’s generalization ability. In summary, our initial exploration into the usage of XAI for enhancing semantic-based test generation has proved very promising. By carefully selecting and applying the most appropriate XAI methods, it is possible to significantly improve the efficiency and effectiveness of the testing process of complex DL systems.

## 7 Related Work

With the increasing application of DL to safety-critical domains such as autonomous driving, XAI algorithms represent one of the default options to debug the predictions and failures of a DL system. In this section, we focus on the main related propositions.

### 7.1 Test Generation for Deep Learning Systems

The three main families of DL test generation are semantic-based input representation, raw input manipulation, and latent space manipulation.

Semantic Input Manipulation (SIM) techniques leverage a semantic representation of the input domain (e.g., a model) to generate test inputs, similar to conventional model-driven engineering practices that uphold compliance with domain-specific constraints [1, 5, 6, 27, 64, 72]. The manipulation occurs on the model, which is subsequently reconverted to the original format [51]. SIM techniques operate within a restricted input space, specifically the control parameters of the model representation. These techniques enhance the realism of the produced outputs by implementing appropriate model constraints.

Several search-based SIM approaches have been applied to DL-based image classifiers. DeepHyperion [104] uses the MAP-Elites Illumination Search algorithm [61] to explore the feature space of the input domain and identify misbehavior-inducing features. DeepMetis [70] a SIM approach that generates inputs that behave correctly on original DL models and misbehave on mutants obtained through injection of realistic faults [34], which can be useful to enhance the mutation killing ability of a test set. DeepJanus [72] is the SIM approach most related to this work since it performs model-based testing of DL systems. Therefore, we performed an explicit empirical comparison with the DeepJanus approach in this work.

To the best of our knowledge, no XAI information is used in existing semantic-based tools to guide test generation. As such, they can be used in conjunction with XMUTANT to improve their effectiveness.

Raw input manipulation (RIM) techniques involve modifying an image’s original pixel space to create a new input by perturbing the pixel values. RIM techniques aim to produce minimal, often imperceptible changes to original to trigger misbehaviors in the DL system. These methods target different aspects of testing, such as data augmentation or adversarial attacks, which are not directly aligned with our goal. Our method is a *functional* test generator, differing from adversarial testing in both goals and techniques. Functional testing creates new, valid, in-distribution inputs to evaluate a DNN’s generalization. In contrast, adversarial testing adds minor perturbations to original inputs to test *robustness*. However, for completeness, we describe the main propositions next.

DeepXplore [67] employs various techniques, including occlusion, light manipulation, and blackout to cause misbehaviors. These perturbations are intended to improve neuron coverage within the DL system. DLFuzz [32] introduces noise to the seed image to increase the likelihood of system misbehavior. DLFuzz generates adversarial inputs for DL systems without relying on cross-referencing other similar DL systems or manual labeling. DeepTest [86] alters the images using synthetic affine transformation from the computer vision domain, such as blurring and brightness adjustments, to create simulated rain/fog effects.

Differently, our technique targets functional testing, specifically testing for generalization of DL systems. We achieve this by using XAI guidance in the pixel space, and link it to the model input to generate inputs beyond the original datasets, while remaining within the same distribution. In [Section 4](#), we have showed that our technique generates more natural samples that adhere to the original data’s manifold, whereas these RIM methods, despite their minimal pixel perturbations, artificially inflate failure rates via distribution shifts.

Latent space manipulation (LIM) techniques generate new inputs by learning and reconstructing the underlying distribution of the input data. Sinvad [42] constructs the input space using Variational Autoencoders (VAE) [16, 48] and navigates the latent space by adding a random value sampled from a normal distribution to a single element of the latent vector. Sinvad aims to explore the latent space by maximizing either the probability of misbehaviors, estimated from the softmax layer output, or by surprise coverage [46]. The Feature Perturbations technique [20, 21] involves injecting perturbations into the output of the generative model’s first layers, which represent high-level features of images. These perturbations can affect various characteristics of the image, such as shape, location, texture, or color. DeepRoad [99] generates driving images using Generative Adversarial Networks (GANs) [28, 29] for image-to-image translation. CIT4DNN [17] combines VAE and combinatorial testing [13]. This allows the systematic exploration and generation of diverse and infrequent input datasets. CIT4DNN partitions latent spaces to create test sets that contain a wide range of feature combinations and rare occurrences. A recently proposed technique, Instance Space Analysis, aims to pinpoint the critical features of test scenarios that impact the detection of unsafe behavior [64].

These aforementioned LIM techniques require large-scale training data to capture complete feature distributions, which inherently limits their applicability in resource-constrained or open-world settings. For non-stationary data streams (e.g., evolving NLP token distributions or evolving self-driving scenarios), LIM requires frequent retraining to accommodate changes in the distribution, which incurs excessive computational costs. Indeed, existing testing propositions are limited to DL systems that take as input individual images [4, 17, 59, 64, 99]. Differently from these LIM approaches, XMUTANT leverages the local explanation from existing seeds to guide model-based test generation, by perturbing semantic representations instead of latent vectors from a learned manifold. In our paper, we showed that the semantic-based representations made XMUTANT applicable across different case studies, both at the model and at the system level.

## 7.2 XAI for DL Testing

Zohdinasab et al. [106] compare three state-of-the-art techniques for explanation of DL failures. They show that local and global XAI techniques provide dissimilar explanations for the same inputs and further research is needed to produce better explanations. VisualBackProp [8] was created to visualize which group of pixels of the input image contributes more to the predictions of a convolutional neural network. Kim and Canny [45] explore the use of heatmaps for explaining the CNN behavior in an ADS. Xu et al. [95] investigated the use of XAI techniques to detect action-inducing objects, i.e., objects that have a relevant effect on a driving decision, and jointly predict actions and their respective explanations. ThirdEye [82] focuses on failure prediction of lane-keeping autonomous driving systems during hazardous driving conditions. They turn heatmaps into confidence scores that are used to discriminate safe from unsafe driving behaviors. The intuition is that uncommon heatmaps are associated with unexpected runtime conditions. In this work we use ThirdEye, equipped with the heatmap derivative configuration, to provide such scores during system-level testing.

Fahmy et al. [23] apply clustering to heatmaps capturing the relevance of the DNN predictions to automatically support the identification of failure-inducing inputs. Such data is used for the retraining of a gaze detection system that uses DNNs to determine the gaze direction of the driver. The authors present an extension of the previous work [22] in which inputs identified by the heatmap-based mechanism are given in input to a search-based test generator.

In contrast, in this work, we use local explanations to support the early detection of corner case inputs of a DL model. In our work we experiment with different XAI algorithms, showing that the choice of the best algorithm is domain and testing level dependent.

## 8 Conclusions

In this work, we describe and evaluate XMUTANT, a semantic-based fuzzer for DL systems that generates inputs that focus on the attention of the system under test through mutations that are informed by the local explanation available from XAI algorithms. We evaluated XMUTANT on both model-level and system-level testing. Our empirical studies show that XMUTANT is significantly more effective and efficient than the state-of-the-art test generation approaches, while preserving a high validity rate of failing test inputs.

## 9 Data Availability

All our results, the source code, and the simulator are accessible and can be reproduced [68].

## 10 Acknowledgments

Matteo Biagiola is partially supported by Fondo Istituzionale per la Ricerca granted by Università della Svizzera italiana (USI).



## 11 Appendix

### 11.1 Effect of Window Size

To validate the choice of the window size  $ws = 3$  for the XMUTANT equipped with Square-Window in MNIST study, we conducted an ablation study across different values (1, 3, 5, 7), as shown in Figure 9. We report the failure rates using both SmoothGrad and Grad-CAM++, for 2000 seeds and with the budget of 200 iterations.

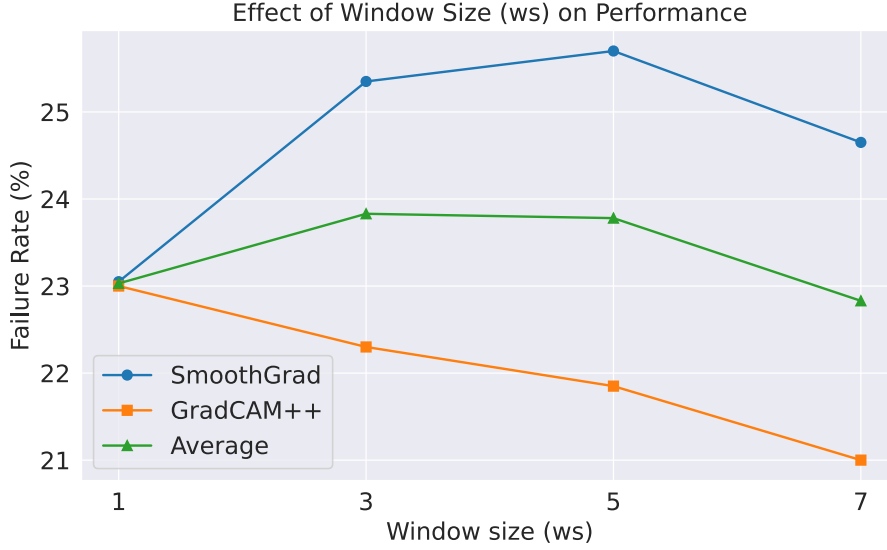


Fig. 9: Effect on Window Size

While SmoothGrad peaks around  $ws = 5$ , Grad-CAM++ performs best at  $ws = 1$ . On average,  $ws = 3$  yields the highest combined score across methods. However, the variation is relatively minor (within 2%), suggesting the method is not overly sensitive to this parameter. Based on this, we selected  $ws = 3$  for running our experiments to balance performance and generality.

### 11.2 Comprehensive result for RQ<sub>1</sub>, RQ<sub>2</sub>, and RQ<sub>3</sub>

Table 5 comprehensively presents the effectiveness, performance, and comparative results for all configurations of XMUTANT and DeepJanus as the baseline, for all case studies (IMDB, MNIST, and ADAS). The table reports the information at different iteration intervals, respectively, 10 for IMDB, 100 for MNIST, and 5 for ADAS. For each configuration of XMUTANT and DeepJanus, the first number denotes the cumulative failure rate, whereas the second number indicates the relative efficiency with respect to the baseline.

Table 5: Detailed results for RQ<sub>1</sub>, RQ<sub>2</sub> and RQ<sub>3</sub>. For each configuration, the two values indicate the *Cumulative Failure Rate [%]/ Relative Efficiency* over different Iterations. P-values indicating statistically significant differences are highlighted in bold.

	Iterations	SmoothGrad		LIME		Integrated Gradients			DJ		
IMDB	10	14/1.93		11/1.44		16/2.05			7/-		
	20	22/2.08		32/2.31		31/2.71			10/-		
	30	31/2.08		42/2.74		46/2.92			14/-		
	40	39/2.12		51/2.82		57/3.05			18/-		
	50	41/2.12		60/2.86		64/3.09			21/-		
	60	47/2.08		63/2.83		67/3.04			24/-		
	70	49/2.03		66/2.75		70/2.94			27/-		
	80	51/1.98		70/2.69		74/2.87			29/-		
	90	53/1.93		72/2.64		76/2.80			31/-		
	100	56/1.90		76/2.74		76/2.59			32/-		
RQ <sub>1</sub>	p-value effect size	6.93E-18 large		1.17E-17 large		7.90E-18 large			-		
RQ <sub>2</sub>	p-value effect size	2.53E-04 medium		1.20E-06 large		2.92E-06 large			-		
		SmoothGrad			Grad-CAM++			Integrated Gradients			DJ
		Clustering		SW	Clustering		SW	Clustering		SW	-
	Iterations	Attractor	Random	Random	Attractor	Random	Random	Attractor	Random	Random	-
MNIST	100	44/7.00	11/1.83	11/2.11	36/4.81	07/1.03	09/1.32	34/5.54	13/2.06	13/2.15	06/-
	200	77/6.37	24/1.77	24/1.82	74/5.54	21/1.32	22/1.48	60/4.92	27/2.00	26/1.97	15/-
	300	88/4.97	38/1.62	36/1.60	88/4.61	35/1.36	34/1.41	74/3.92	41/1.80	37/1.72	26/-
	400	94/4.03	47/1.52	45/1.47	94/3.84	46/1.34	45/1.36	80/3.26	51/1.67	47/1.57	35/-
	500	96/3.42	55/1.44	53/1.39	97/3.30	54/1.32	54/1.32	85/2.82	59/1.57	54/1.47	43/-
	600	97/2.99	62/1.38	60/1.33	98/2.91	60/1.28	61/1.29	87/2.51	66/1.49	61/1.39	51/-
	700	98/2.67	67/1.32	65/1.28	98/2.61	66/1.25	67/1.26	89/2.27	71/1.43	66/1.33	58/-
	800	98/2.43	71/1.28	69/1.25	99/2.39	70/1.22	73/1.24	90/2.10	76/1.38	71/1.29	62/-
	900	99/2.26	75/1.25	73/1.22	99/2.23	74/1.20	77/1.22	91/1.96	79/1.35	74/1.26	67/-
	1000	99/2.12	78/1.23	76/1.20	99/2.09	77/1.19	80/1.21	92/1.86	82/1.32	77/1.23	70/-
RQ <sub>1</sub>	p-value effect size	4.94E-165 large	1.09E-164 small	3.46E-165 small	2.57E-164 large	1.19E-161 small	2.10E-164 small	4.86E-165 large	7.03E-165 small	7.05E-165 small	-
RQ <sub>2</sub>	p-value effect size	1.75E-183 large	2.47E-15 small	1.59E-10 small	4.46E-188 large	1.55E-10 small	2.55E-08 negligible	2.06E-142 large	2.05E-25 small	3.91E-15 small	-
		SmoothGrad			Grad-CAM++			Integrated Gradients			DJ
	Iterations	Random	Low	High	Random	Low	High	Random	Low	High	-
ADAS	5	27/2.00	31/2.20	42/2.87	49/2.80	42/2.53	36/2.07	45/2.13	36/2.53	40/2.60	22/-
	10	65/1.35	60/1.36	67/1.69	75/1.86	87/1.78	75/1.60	64/1.57	65/1.60	58/1.56	51/-
	15	84/1.33	76/1.25	82/1.45	85/1.58	93/1.65	95/1.54	75/1.38	78/1.37	80/1.38	65/-
	20	89/1.31	82/1.21	84/1.36	95/1.47	93/1.53	95/1.48	87/1.30	85/1.30	85/1.32	75/-
	25	91/1.25	85/1.16	93/1.29	95/1.39	95/1.42	95/1.39	91/1.25	87/1.23	89/1.25	84/-
	30	91/1.21	89/1.13	95/1.24	95/1.32	95/1.35	95/1.32	95/1.21	87/1.18	89/1.20	84/-
	35	93/1.18	89/1.11	95/1.21	95/1.27	95/1.29	95/1.27	95/1.19	91/1.15	91/1.17	89/-
	40	95/1.15	89/1.09	95/1.18	95/1.23	95/1.25	95/1.23	95/1.16	93/1.12	95/1.14	95/-
RQ <sub>1</sub>	p-value effect size	8.23E-08 small	5.26E-06 small	1.05E-07 small	1.07E-07 medium	1.10E-07 medium	1.09E-07 medium	1.05E-07 small	1.54E-07 small	1.07E-07 small	-
RQ <sub>2</sub>	p-value effect size	9.62E-03 small	2.87E-01 -	6.93E-03 small	1.77E-04 medium	1.33E-04 medium	2.19E-04 medium	6.05E-02 -	5.33E-02 -	1.47E-02 -	-

### 11.3 Sanity Check on LLM's Validation

We conducted a sanity check of ChatGPT's output by randomly sampling 18 generated test cases across different configurations. Each case included the modified input, ChatGPT's sentiment prediction, and its explanation. Here is an example of generated input with the labels provided by SUT and ChatGPT's validation.

Generated Text:

"Unreal and contrived melodrama, with a screenplay altered and altered by Mel from his play, The Man Business Organization. A room put up and improving <unk>, taking atomic and atomic amount il, a troubled and trouble oneself <unk>, who may be homicidal and murderous..."

Expected Label: "Negative",

Predicted Label: "Positive",

ChatGPT's Answers: "negative. The review expresses a strong sense of

dissatisfaction with the film, using words like 'unreal', 'contrived'... These terms indicate a negative sentiment towards the movie, suggesting that the reviewer found it lacking in quality and engaging elements."

5 human annotators independently assessed whether (i) the predicted sentiment aligned with the input and (ii) the explanation was logically consistent and specific. To quantify the overall agreement among human participants and Chat-GPT, we computed Fleiss' kappa, obtaining a value of 0.71, indicating a substantial inter-rater agreement [25, 50]. In addition to the group-level measure, we assessed the individual consistency of each participant by comparing their responses with those of the others on every task. The results reveal that the assessment of Chat-GPT was the most reliable (0.88) with the highest agreement rate, closely followed by other human annotators (0.78-0.88).

## References

1. Abdessalem, R.B., Panichella, A., Nejati, S., Briand, L.C., Stifter, T.: Testing autonomous cars for feature interaction failures using many-objective search. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, pp. 143–154. ACM, New York, NY, USA (2018). DOI 10.1145/3238147.3238192. URL <http://doi.acm.org/10.1145/3238147.3238192>
2. Adebayo, J., Gilmer, J., Muelly, M., Goodfellow, I., Hardt, M., Kim, B.: Sanity checks for saliency maps. *Advances in neural information processing systems* **31** (2018)
3. Amini, M.H., Naseri, S., Nejati, S.: Evaluating the impact of flaky simulators on testing autonomous driving systems. *Empirical Softw. Engg.* **29**(2) (2024). DOI 10.1007/s10664-023-10433-5. URL <https://doi.org/10.1007/s10664-023-10433-5>
4. Baresi, L., Hu, D.Y.X., Stocco, A., Tonella, P.: Efficient domain augmentation for autonomous driving testing using diffusion models. In: Proceedings of 47th International Conference on Software Engineering, ICSE '25. IEEE (2025)
5. Ben Abdessalem, R., Nejati, S., Briand, L.C., Stifter, T.: Testing advanced driver assistance systems using multi-objective search and neural networks. In: 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 63–74 (2016)
6. Ben Abdessalem, R., Nejati, S., C. Briand, L., Stifter, T.: Testing vision-based control systems using learnable evolutionary algorithms. In: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), pp. 1016–1026 (2018). DOI 10.1145/3180155.3180160
7. Biagiola, M., Stocco, A., Riccio, V., Tonella, P.: Two is better than one: Digital siblings to improve autonomous driving testing. *Empirical Software Engineering* (2024)
8. Bojarski, M., Choromanska, A., Choromanski, K., Firner, B., Jackel, L., Muller, U., Zieba, K.: Visualbackprop: efficient visualization of cnns (2016). DOI 10.48550/ARXIV.1611.05418. URL <https://arxiv.org/abs/1611.05418>
9. Bojarski, M., Testa, D.D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., Zieba, K.: End to end learning for self-driving cars. *CoRR abs/1604.07316* (2016). URL <http://arxiv.org/abs/1604.07316>
10. Catmull, E., Rom, R.: A class of local interpolating splines. In: Computer aided geometric design, pp. 317–326. Elsevier (1974)
11. Chattopadhyay, A., Sarkar, A., Howlader, P., Balasubramanian, V.N.: Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In: 2018 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE (2018). DOI 10.1109/wacv.2018.00097. URL <http://dx.doi.org/10.1109/WACV.2018.00097>
12. Cheng, M., Zhou, Y., Xie, X.: Behavexplor: Behavior diversity guided testing for autonomous driving systems. In: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2023, p. 488–500. Association for Computing Machinery, New York, NY, USA (2023). DOI 10.1145/3597926.3598072. URL <https://doi.org/10.1145/3597926.3598072>
13. Cohen, D., Dalal, S., Fredman, M., Patton, G.: The aetg system: an approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering* **23**(7), 437–444 (1997). DOI 10.1109/32.605761
14. Cohen, J.: Statistical power analysis for the behavioral sciences. L. Erlbaum Associates, Hillsdale, N.J (1988)
15. Deng, L.: The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* **29**(6), 141–142 (2012)
16. Doersch, C.: Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908* (2016)
17. Dola, S., McDaniel, R., Dwyer, M.B., Soffa, M.L.: Cit4dnn: Generating diverse and rare inputs for neural networks using latent space combinatorial testing. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, pp. 1–13 (2024)
18. Došilović, F.K., Brčić, M., Hlupić, N.: Explainable artificial intelligence: A survey. In: 2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO), pp. 0210–0215. IEEE (2018)
19. Du, M., Liu, N., Hu, X.: Techniques for interpretable machine learning. *Communications of the ACM* **63**(1), 68–77 (2019)
20. Dunn, I., Melham, T., Kroening, D.: Semantic adversarial perturbations using learnt representations. *CoRR abs/2001.11055* (2020). URL <https://arxiv.org/abs/2001.11055>

21. Dunn, I., Pouget, H., Kroening, D., Melham, T.: Exposing previously undetectable faults in deep neural networks. In: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 56–66 (2021)
22. Fahmy, H., Pastore, F., Briand, L., Stifter, T.: Simulator-based explanation and debugging of hazard-triggering events in dnn-based safety-critical systems. *ACM Transactions on Software Engineering and Methodology* **32**(4), 1–47 (2023). DOI 10.48550/ARXIV.2204.00480. URL <https://arxiv.org/abs/2204.00480>
23. Fahmy, H.M., Bagherzadeh, M., Pastore, F., Briand, L.C.: Supporting DNN safety analysis and retraining through heatmap-based unsupervised learning. *CoRR abs/2002.00863* (2020). URL <https://arxiv.org/abs/2002.00863>
24. Farin, G.: Algorithms for rational bézier curves. *Computer-aided design* **15**(2), 73–77 (1983)
25. Fleiss, J.L.: Measuring nominal scale agreement among many raters. *Psychological bulletin* **76**(5), 378 (1971)
26. Fong, R.C., Vedaldi, A.: Interpretable explanations of black boxes by meaningful perturbation. In: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 3449–3457 (2017). DOI 10.1109/ICCV.2017.371
27. Gambi, A., Mueller, M., Fraser, G.: Automatically testing self-driving cars with search-based procedural content generation. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, pp. 318–328. ACM, New York, NY, USA (2019). DOI 10.1145/3293882.3330566. URL <http://doi.acm.org/10.1145/3293882.3330566>
28. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. *Advances in neural information processing systems* **27**, 2672–2680 (2014). URL <http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf>
29. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks. *Communications of the ACM* **63**(11), 139–144 (2020)
30. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. *CoRR abs/1412.6572* (2014). URL <https://api.semanticscholar.org/CorpusID:6706414>
31. Gunning, D., Stefik, M., Choi, J., Miller, T., Stumpf, S., Yang, G.Z.: Xai—explainable artificial intelligence. *Science robotics* **4**(37), eaay7120 (2019)
32. Guo, J., Jiang, Y., Zhao, Y., Chen, Q., Sun, J.: 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, pp. 739–743 (2018)
33. Hochreiter, S.: Long short-term memory. *Neural Computation* MIT-Press (1997)
34. Humberova, N., Jahangirova, G., Bavota, G., Riccio, V., Stocco, A., Tonella, P.: Taxonomy of real faults in deep learning systems. In: Proceedings of 42nd International Conference on Software Engineering, ICSE ’20, p. 12 pages. ACM, New York, NY, USA (2020). DOI 10.1145/3377811.3380395
35. Hussain, M., Ali, N., Hong, J.E.: Deepguard: A framework for safeguarding autonomous driving systems from inconsistent behaviour. *Automated Software Engg.* **29**(1) (2022). DOI 10.1007/s10515-021-00310-0. URL <https://doi.org/10.1007/s10515-021-00310-0>
36. ISO: Road vehicles – Functional safety (2011)
37. Jahangirova, G., Stocco, A., Tonella, P.: Quality metrics and oracles for autonomous vehicles testing. In: Proceedings of 14th IEEE International Conference on Software Testing, Verification and Validation, ICST ’21. IEEE (2021)
38. Jetley, S., Lord, N.A., Lee, N., Torr, P.H.: Learn to pay attention. *arXiv preprint arXiv:1804.02391* (2018)
39. Jha, S., Banerjee, S.S., Tsai, T., Hari, S.K.S., Sullivan, M.B., Kalbarczyk, Z.T., Keckler, S.W., Iyer, R.K.: ML-based fault injection for autonomous vehicles: A case for bayesian fault injection. 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) pp. 112–124 (2019). URL <https://api.semanticscholar.org/CorpusID:195776612>
40. Jiang, Z., Li, H., Wang, R., Tian, X., Liang, C., Yan, F., Zhang, J., Liu, Z.: Validity matters: Uncertainty-guided testing of deep neural networks. *Software Testing, Verification and Reliability* p. e1894 (2024)

41. Julian, K.D., Kochenderfer, M.J., Owen, M.P.: Deep neural network compression for aircraft collision avoidance systems. *CoRR* **abs/1810.04240** (2018)
42. Kang, S., Feldt, R., Yoo, S.: 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*, pp. 521–528 (2020)
43. Kang, S., Feldt, R., Yoo, S.: Deceiving humans and machines alike: Search-based test input generation for dnns using variational autoencoders. *ACM Transactions on Software Engineering Methodologies* **33**, 103:1–24 (2024). DOI 10.1145/3635706
44. Kao, V.: Sentimental analysis on imdb by lstm (2018). URL <https://www.kaggle.com/code/vincentman0403/sentimental-analysis-on-imdb-by-lstm/notebook>. Accessed: 2024-08-18
45. Kim, J., Canny, J.: Interpretable learning for self-driving cars by visualizing causal attention (2017). DOI 10.48550/ARXIV.1703.10631. URL <https://arxiv.org/abs/1703.10631>
46. Kim, J., Feldt, R., Yoo, S.: Guiding deep learning system testing using surprise adequacy. In: *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), ICSE '19*, pp. 1039–1049. IEEE, IEEE Press, Piscataway, NJ, USA (2019). DOI 10.1109/ICSE.2019.00108. URL <https://doi.org/10.1109/ICSE.2019.00108>
47. Kim, S., Liu, M., Rhee, J.J., Jeon, Y., Kwon, Y., Kim, C.H.: DriveFuzz. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM (2022). DOI 10.1145/3548606.3560558. URL <https://doi.org/10.1145/3548606.3560558>
48. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013)
49. Kothari, V.: Image classification of mnist using VGG16. *Kaggle Notebook* (2020). URL <https://www.kaggle.com/code/viratkothari/image-classification-of-mnist-using-vgg16>
50. Landis, J.R., Koch, G.G.: The measurement of observer agreement for categorical data. *Biometrics* pp. 159–174 (1977)
51. Larman, C., et al.: *Applying UML and patterns, vol. 2*. Prentice Hall Upper Saddle River (1998)
52. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86**(11), 2278–2324 (1998)
53. Li, G., Li, Y., Jha, S., Tsai, T., Sullivan, M., Hari, S.K.S., Kalbarczyk, Z., Iyer, R.: Av-fuzzer: Finding safety violations in autonomous driving systems. In: *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pp. 25–36 (2020). DOI 10.1109/ISSRE5003.2020.00012
54. Li, X.H., Shi, Y., Li, H., Bai, W., Cao, C.C., Chen, L.: An experimental study of quantitative evaluations on saliency methods. In: *Proceedings of the 27th ACM sigkdd conference on knowledge discovery & data mining*, pp. 3200–3208 (2021)
55. Lou, G., Deng, Y., Zheng, X., Zhang, M., Zhang, T.: Testing of autonomous driving systems: Where are we and where should we go? (2021). DOI 10.48550/ARXIV.2106.12233. URL <https://arxiv.org/abs/2106.12233>
56. Lu, C., Shi, Y., Zhang, H., Zhang, M., Wang, T., Yue, T., Ali, S.: Learning configurations of operating environment of autonomous vehicles to maximize their collisions. *IEEE Transactions on Software Engineering* **49**(1), 384–402 (2023). DOI 10.1109/TSE.2022.3150788
57. Ma, L., Juefei-Xu, F., Zhang, F., Sun, J., Xue, M., Li, B., Chen, C., Su, T., Li, L., Liu, Y., Zhao, J., Wang, Y.: Deepgauge: Multi-granularity testing criteria for deep learning systems. In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018*, pp. 120–131. ACM, New York, NY, USA (2018). DOI 10.1145/3238147.3238202. URL <http://doi.acm.org/10.1145/3238147.3238202>
58. Maas, A.L., Daly, R.E., Pham, P.T., Huang, D., Ng, A.Y., Potts, C.: Learning word vectors for sentiment analysis. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1, HLT '11*, pp. 142–150. Association for Computational Linguistics, Stroudsburg, PA, USA (2011). URL <http://dl.acm.org/citation.cfm?id=2002472.2002491>
59. Maryam, Biagiola, M., Stocco, A., Riccio, V.: Benchmarking Generative AI Models for Deep Learning Test Input Generation. In: *Proceedings of 18th IEEE International Conference on Software Testing, Verification and Validation, ICST '25*, p. 12 pages. IEEE (2025)

60. Miller, G.A.: Wordnet: a lexical database for english. *Communications of the ACM* **38**(11), 39–41 (1995)
61. Mouret, J., Clune, J.: Illuminating search spaces by mapping elites. *CoRR abs/1504.04909* (2015). URL <http://arxiv.org/abs/1504.04909>
62. Mullins, G.E., Stankiewicz, P.G., Hawthorne, R.C., Gupta, S.K.: Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles. *Journal of Systems and Software* **137**, 197–215 (2018). DOI <https://doi.org/10.1016/j.jss.2017.10.031>. URL <http://www.sciencedirect.com/science/article/pii/S0164121217302546>
63. Naeem, M.F., Oh, S.J., Uh, Y., Choi, Y., Yoo, J.: Reliable fidelity and diversity metrics for generative models (2020)
64. Neelofar, N., Aleti, A.: Identifying and explaining safety-critical scenarios for autonomous vehicles via key features. *ACM Transactions on Software Engineering and Methodology* **33**(4), 1–32 (2024)
65. OpenAI: Gpt-4o mini: advancing cost-efficient intelligence. <https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/> (2024). Accessed: 2024-09-08
66. OpenAI: The most powerful platform for building ai products. <https://openai.com/api/> (2024). Accessed: 2024-09-08
67. Pei, K., Cao, Y., Yang, J., Jana, S.: Deepxplore: Automated whitebox testing of deep learning systems. In: *proceedings of the 26th Symposium on Operating Systems Principles, SOSP '17*, vol. 62, pp. 1–18. ACM, New York, NY, USA (2017). DOI [10.1145/3132747.3132785](https://doi.org/10.1145/3132747.3132785). URL <http://doi.acm.org/10.1145/3132747.3132785>
68. Replication package. <https://github.com/ast-fortiss-tum/XMutant> (2025)
69. Ribeiro, M.T., Singh, S., Guestrin, C.: "why should I trust you?": Explaining the predictions of any classifier. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 1135–1144 (2016)
70. Riccio, V., Humbatova, N., Jahangirova, G., Tonella, P.: Deepmetis: Augmenting a deep learning test set to increase its mutation score. In: *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering, ASE '21. IEEE/ACM* (2021)
71. Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M., Tonella, P.: Testing Machine Learning based Systems: A Systematic Mapping. *Empirical Software Engineering* (2020)
72. Riccio, V., Tonella, P.: Model-Based Exploration of the Frontier of Behaviours for Deep Learning System Testing. In: *Proceedings of ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE '20* (2020)
73. Riccio, V., Tonella, P.: When and why test generators for deep learning produce invalid inputs: an empirical study. In: *Proceedings of 45th International Conference on Software Engineering, ICSE '23*, p. 12 pages. ACM (2023)
74. Samek, W., Montavon, G., Vedaldi, A., Hansen, L.K., Müller, K.R.: Explainable AI: interpreting, explaining and visualizing deep learning, vol. 11700. Springer Nature (2019)
75. Samek, W., Wiegand, T., Müller, K.R.: Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296* (2017)
76. Schütze, H., Manning, C.D., Raghavan, P.: Introduction to information retrieval, vol. 39. Cambridge University Press Cambridge (2008)
77. Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Grad-cam: Visual explanations from deep networks via gradient-based localization. In: *Proceedings of the IEEE international conference on computer vision*, pp. 618–626 (2017)
78. Shannon, C.E.: A mathematical theory of communication. *The Bell system technical journal* **27**(3), 379–423 (1948)
79. Shrikumar, A., Greenside, P., Kundaje, A.: Learning important features through propagating activation differences. In: *International conference on machine learning*, pp. 3145–3153. PMLR (2017)
80. Simonyan, K., Vedaldi, A., Zisserman, A.: Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034* (2013)
81. Smilkov, D., Thorat, N., Kim, B., Viégas, F., Wattenberg, M.: Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825* (2017)
82. Stocco, A., Nunes, P.J., d'Amorim, M., Tonella, P.: ThirdEye: Attention Maps for Safe Autonomous Driving Systems. In: *Proceedings of 37th IEEE/ACM International Conference on Automated Software Engineering, ASE '22. IEEE/ACM* (2022). DOI [10.1145/3551349.3556968](https://doi.org/10.1145/3551349.3556968). Accepted



83. Stocco, A., Weiss, M., Calzana, M., Tonella, P.: Misbehaviour prediction for autonomous driving systems. In: Proceedings of 42nd International Conference on Software Engineering, ICSE '20, p. 12 pages. ACM (2020)
84. Sundararajan, M., Taly, A., Yan, Q.: Axiomatic attribution for deep networks (2017)
85. Tang, S., Zhang, Z., Zhang, Y., Zhou, J., Guo, Y., Liu, S., Guo, S., Li, Y., Ma, L., Xue, Y., Liu, Y.: A survey on automated driving system testing: Landscapes and trends. CoRR **abs/2206.05961** (2022). DOI 10.48550/arXiv.2206.05961. URL <https://arxiv.org/abs/2206.05961>
86. Tian, Y., Pei, K., Jana, S., Ray, B.: Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In: Proceedings of the 40th International Conference on Software Engineering, ICSE '18, pp. 303–314. ACM, New York, NY, USA (2018). DOI 10.1145/3180155.3180220. URL <http://doi.acm.org/10.1145/3180155.3180220>
87. Tjoa, E., Khok, H.J., Chouhan, T., Guan, C.: Improving deep neural network classification confidence using heatmap-based explainable AI. CoRR **abs/2201.00009** (2022). URL <https://arxiv.org/abs/2201.00009>
88. Udacity: A self-driving car simulator built with Unity. <https://github.com/udacity/self-driving-car-sim> (2017). Online; accessed 18 August 2019
89. Unity3d. <https://unity.com> (2021)
90. Vilone, G., Longo, L.: Explainable artificial intelligence: a systematic review. arXiv preprint arXiv:2006.00093 (2020)
91. Wang, R., Guo, J., Gao, C., Fan, G., Chong, C.Y., Xia, X.: Can llms replace human evaluators? an empirical study of llm-as-a-judge in software engineering (2025). URL <https://arxiv.org/abs/2502.06193>
92. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics Bulletin **1**(6), 80 (1945). DOI 10.2307/3001968. URL <https://doi.org/10.2307/3001968>
93. Wu, J.: Advances in K-means clustering: a data mining thinking. Springer Science & Business Media (2012)
94. Xie, X., Ma, L., Juefei-Xu, F., Xue, M., Chen, H., Liu, Y., Zhao, J., Li, B., Yin, J., See, S.: Deephunter: A coverage-guided fuzz testing framework for deep neural networks. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA '19, pp. 146–157. ACM, New York, NY, USA (2019). DOI 10.1145/3293882.3330579. URL <http://doi.acm.org/10.1145/3293882.3330579>
95. Xu, Y., Yang, X., Gong, L., Lin, H.C., Wu, T.Y., Li, Y., Vasconcelos, N.: Explainable object-induced action decision for autonomous vehicles (2020). DOI 10.48550/ARXIV.2003.09405. URL <https://arxiv.org/abs/2003.09405>
96. Yuan, Y., Pang, Q., Wang, S.: You can't see the forest for its trees: Assessing deep neural network testing via neural coverage. CoRR **abs/2112.01955** (2021). URL <https://arxiv.org/abs/2112.01955>
97. Zhang, C., Almpandis, G., Fan, G., Deng, B., Zhang, Y., Liu, J., Kamel, A., Soda, P., Gama, J.: A systematic review on long-tailed learning (2024). URL <https://arxiv.org/abs/2408.00483>
98. Zhang, J., Keung, J., Ma, X., Li, X., Xiao, Y., Li, Y., Chan, W.K.: Enhancing valid test input generation with distribution awareness for deep neural networks. In: 2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC), pp. 1095–1100. IEEE (2024)
99. Zhang, M., Zhang, Y., Zhang, L., Liu, C., Khurshid, S.: 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, ASE 2018, pp. 132–142. ACM, New York, NY, USA (2018). DOI 10.1145/3238147.3238187. URL <http://doi.acm.org/10.1145/3238147.3238187>
100. Zhang, Q., Wang, H., Lu, H., Won, D., Yoon, S.W.: Medical image synthesis with generative adversarial networks for tissue recognition. In: 2018 IEEE International Conference on Healthcare Informatics (ICHI). IEEE (2018)
101. Zhao, J., Feng, Q., Wu, P., Lupu, R.A., Wilke, R.A., Wells, Q.S., Denny, J.C., Wei, W.Q.: Learning from longitudinal data in electronic health record and genetic data to improve cardiovascular event prediction. Scientific Reports **9**(1), 717 (2019). DOI 10.1038/s41598-018-36745-x. URL <https://doi.org/10.1038/s41598-018-36745-x>
102. Zheng, L., Chiang, W.L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E.P., Zhang, H., Gonzalez, J.E., Stoica, I.: Judging llm-as-a-judge with mt-bench and chatbot arena (2023). URL <https://arxiv.org/abs/2306.05685>

103. Zhong, Z., Kaiser, G., Ray, B.: Neural network guided evolutionary fuzzing for finding traffic violations of autonomous vehicles (2021)
104. Zohdinasab, T., Riccio, V., Gambi, A., Tonella, P.: 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, ISSTA '21, pp. 79–90. Association for Computing Machinery (2021)
105. Zohdinasab, T., Riccio, V., Tonella, P.: Deepatash: Focused test generation for deep learning systems. In: R. Just, G. Fraser (eds.) Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2023, Seattle, WA, USA, July 17–21, 2023, pp. 954–966. ACM (2023). DOI 10.1145/3597926.3598109. URL <https://doi.org/10.1145/3597926.3598109>
106. Zohdinasab, T., Riccio, V., Tonella, P.: An empirical study on low- and high-level explanations of deep learning misbehaviours. In: 2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1–11 (2023). DOI 10.1109/ESEM56168.2023.10304866