Simulator-based explanation and debugging of hazard-triggering events in DNN-based safety-critical systems

HAZEM FAHMY, SnT Centre, University of Luxembourg, Luxembourg

FABRIZIO PASTORE, SnT Centre, University of Luxembourg, Luxembourg

LIONEL BRIAND, SnT Centre, University of Luxembourg, Luxembourg and School of EECS, University of Ottawa, Canada

THOMAS STIFTER, IEE S.A., Luxembourg

When Deep Neural Networks (DNNs) are used in safety-critical systems, engineers should determine the safety risks associated with failures (i.e., erroneous outputs) observed during testing. For DNNs processing images, engineers visually inspect all failure-inducing images to determine common characteristics among them. Such characteristics correspond to hazard-triggering events (e.g., low illumination) that are essential inputs for safety analysis. Though informative, such activity is expensive and error-prone.

To support such safety analysis practices, we propose SEDE, a technique that generates readable descriptions for commonalities in failure-inducing, real-world images and improves the DNN through effective retraining. SEDE leverages the availability of simulators, which are commonly used for cyber-physical systems. It relies on genetic algorithms to drive simulators towards the generation of images that are similar to failure-inducing, real-world images in the test set; it then employs rule learning algorithms to derive expressions that capture commonalities in terms of simulator parameter values. The derived expressions are then used to generate additional images to retrain and improve the DNN.

With DNNs performing in-car sensing tasks, SEDE successfully characterized hazard-triggering events leading to a DNN accuracy drop. Also, SEDE enabled retraining leading to significant improvements in DNN accuracy, up to 18 percentage points.

CCS Concepts: • **Software and its engineering** → *Software maintenance tools*; **Search-based software engineering**; **Software testing and debugging**; • **Computing methodologies** → *Machine learning*.

Additional Key Words and Phrases: DNN Explanation, DNN Functional Safety Analysis, DNN Debugging, Heatmaps, Explainable AI

Authors' addresses: Hazem Fahmy, SnT Centre, University of Luxembourg, JFK 29, Luxembourg, Luxembourg, hazem.fahmy@uni.lu; Fabrizio Pastore, SnT Centre, University of Luxembourg, JFK 29, Luxembourg, Luxembourg, fabrizio.pastore@uni.lu; Lionel Briand, SnT Centre, University of Luxembourg, JFK 29, Luxembourg, Luxembourg, School of EECS, University of Ottawa, Ottawa, Canada, lionel.briand@uni.lu; Thomas Stifter, IEE S.A., Luxembourg, Luxembourg, thomas.stifter@iee.lu.

1 INTRODUCTION

Deep Neural Networks (DNNs) are increasingly common building blocks in many modern software systems, including safety-critical cyber-physical systems. This is true for automotive systems, where DNNs are used for a range of activities, from automating driving tasks, such as emergency braking or lane changing [52, 62], to supporting passenger safety through drowsiness detection and gaze detection systems [50].

The DNNs used in computer-vision components of safety-critical autonomous systems are trained and tested using both images generated with simulators and real-world images. Because of the costs associated with manual labelling and data collection, the availability of real-world images is often limited, which may negatively affect the accuracy of DNNs. As a result, DNN developers are increasingly relying on simulator images to train DNN models, while using real-world images to fine-tune the DNN and then test it [6, 14, 31, 36].

In the presence of DNN failures (i.e., DNN outputs that do not match the ground truth [1]), engineers visually inspect the failure-inducing images to perform root cause analysis. In a safety-critical context, the objective of such root cause analysis is to identify the events that have triggered DNN failures, which are referred to as the *hazard-triggering events*. Indeed, such identification is part of safety standards (e.g., ISO/PAS 21448 Chapter 7 [32]) and enables engineers to evaluate the risk associated to potentially hazardous behaviors of the DNN-based system. For example, when DNN inputs are images, by iteratively inspecting multiple images, engineers should be able to group images presenting common characteristics and therefore identify the hazard-triggering events among such commonalities (e.g., the drivers' head is turned above a certain angle and there is a shadow on their face)[2].

In a safety context, the identification of hazard-triggering events enables engineers to estimate the *probability of exposure* to a specific hazard (e.g., a system failure caused by an erroneous output from the DNN), which is necessary for risk assessment. For our example above, engineers may determine how likely it is for a shadow to partially cover the driver's face while her head is turned.

Hazard-triggering events represent the root causes of DNN failures. Unfortunately, their manual identification is expensive and error-prone because it requires the inspection and comparison of many DNN inputs (e.g., images). To simplify the identification of DNN failures, engineers may rely on visualization techniques to generate heatmaps, i.e., images that use colors to capture the relevance of pixels in their contribution to a DNN output [47, 59]. Although a human operator may determine the root cause of a DNN failure by noticing that multiple heatmaps highlight the same objects (e.g., long hair [59]), the analysis of a large set of failure-inducing images is error-prone (e.g., engineers may not notice some of the failure causes). To overcome this problem, our previous work, Heatmap-based Unsupervised Debugging of DNNs (HUDD) [17], automatically groups images showing a same root cause by analyzing heatmaps derived from DNN outputs and neuron activations — such groups of images are named root cause clusters (RCCs). The rationale behind HUDD is that images sharing the same root causes (e.g., a face turned left) should present similar neuron activations and, consequently, similar heatmaps. However, with HUDD, root cause analysis still remains error-prone because it relies on the capability of the engineer to interpret

---

[1]In this paper, we use the term *DNN output* to refer to either the predicted class, for classifier DNNs, or the predicted numerical value, for DNNs addressing regression problems. For classifier DNNs, we observe a DNN failure when the predicted class does not match the expected one. For regression DNNs, a DNN failure is observed when the difference between the predicted and the actual value is above a threshold specified by a domain expert.

[2]Although hazards are triggered by specific inputs (e.g., head turned 43 degrees leads to misclassification), when identifying hazard-triggering events, engineers are interested in generalizing from the characteristics of multiple failure-inducing inputs (e.g., head turned more than 40 degrees).

the generated outputs (e.g., she may not notice that DNN failures depend not only on a face being turned left but also on a specific illumination angle).

In this paper, we address the problem of automatically generating explicit descriptions for hazard-triggering events from real-world images. Such descriptions are provided in terms of logical expressions constraining the configuration parameters of the simulator used to train the DNN (e.g., rotation angle of the driver's head and illumination angle). For example, we may report that the hazard-triggering event that prevents the gaze angle from being correctly estimated is the driver's head turned by more than 60 degrees with an illumination angle above 45 degrees (i.e., both eyes are barely visible and covered by a shadow). We name our approach Simulator-based Explanations for DNN failurEs (SEDE).

Given a set of failure-inducing real-world images (e.g., the failure-inducing images in the test set), SEDE relies on HUDD to generate RCCs. We assume that all the images belonging to a RCC present the same hazard-triggering events, as suggested by HUDD's empirical results. For each RCC identified by HUDD, SEDE relies on the simulator used for DNN training to generate more images that belong to the RCC. Since a RCC characterizes a small portion of the input space, SEDE relies on evolutionary algorithms to efficiently generate RCC images; indeed, evolutionary algorithms explore the input space guided by fitness functions measuring how close an input is to satisfying a given objective. To better identify the commonalities among RCC images and avoid generating images that present characteristics that are accidentally shared [3], we propose PaiR (Pairwise Replacement), a genetic algorithm that not only generates images belonging to the RCC but also maximizes their diversity.

Once PaiR has generated images belonging to the RCC, to ensure that these images include hazard-triggering events, SEDE produces additional images that are mispredicted, in addition to belonging to the RCC. This is achieved by relying on a modified version of the multi-objective algorithm NSGA-II (hereafter, *NSGA-II'*). Different from recent extensions of NSGA-II [57, 58] that aim to maximize diversity (hereafter, DeepNSGA-II), PaiR does not rely on an archive and does not require the definition of thresholds for selecting the final set of images. Finally, to precisely characterize hazard-triggering events (i.e., to distinguish between commonalities that cause DNN failures and accidental similarities), SEDE relies on one additional execution of *NSGA-II'* to generate additional images that are similar to failure-inducing images but do not cause a DNN failure.

The availability of failing and passing images enables SEDE to derive, leveraging the PART decision rule learning algorithm [19], a set of expressions for the simulator parameters that capture commonalities among failure-inducing images. These expressions characterize hazard-triggering events and also delimit part of the input space that shall be considered unsafe. The expressions generated by SEDE can be used to either estimate the probability of exposure to the hazard-triggering events (based on domain knowledge) or improve the DNN. Improvements in DNN accuracy can be achieved by retraining the model using additional images generated using a simulator, configured with parameters that match the expressions generated by SEDE.

We have performed an empirical evaluation of our approach with three relevant case study subjects provided by our industry partner in the automotive domain, IEE Sensing [29]. Our subjects concern head pose classification and facial landmarks detection, which are important features for in-car sensing solutions such as drowsiness detection. Our empirical results show that (1) PaiR outperforms NSGA-II and DeepNSGA-II in identifying a larger set of diverse images belonging to a higher number of RCCs, (2) SEDE effectively identifies a set of diverse images with common characteristics, (3) SEDE derives expressions that identify unsafe parts of the input space where the

---

[3]An evolutionary algorithm may get stuck in a local optimum and, for example, generate images of persons with blue eyes whose head is turned 45 degrees, though the eye color does not affect the DNN outputs.

DNN accuracy drops up to 80%, and, (4) the expressions generated by SEDE can be used to improve the DNN accuracy (up to 18 percentage points in accuracy).

The rest of the paper is structured as follows. In Section 2, we present the required background (heatmaps generation, HUDD, many-objective search algorithms, and rule learning algorithms). In Section 3, we describe SEDE. In Section 4, we present our empirical evaluation, which includes a comparison with NSGA-II, DeepNSGA-II, HUDD, and a random baseline. In Section 5, we discuss related work. Finally, we conclude this paper in Section 6.

2 BACKGROUND

2.1 DNN Explanation and Heatmaps

Most of the approaches that aim to explain DNN outputs [22] concern the generation of heatmaps that capture the importance of pixels in image predictions. They include both approaches that are black-box (i.e., determine important pixels by observing how the DNN output changes after semi-randomly modifying the input pixel values, without relying on other information [11, 54]) and white-box (i.e., rely on information captured from internal layers [47, 59, 60, 67, 68]).

Among the existing heatmap generation approaches, HUDD relies on the Layer-Wise Relevance Propagation (LRP) [47], a white-box approach. First, different from black-box approaches, LRP assesses the relevance of neurons belonging to internal DNN layers, which enables HUDD to cluster images based on neuron relevance in internal DNN layers. This can be very effective as internal DNN layers act as an abstraction over the inputs and can filter out irrelevant information such as image background. Second, different from white-box approaches that generate class activation maps (CAM) by backpropagating the softmax output to feature maps[4] and upsampling [68], it accounts for all the neurons and layers in the DNN. Third, different from deconvolutional networks [67] and guided backpropagation [60], it generates precise, non-sparse heatmaps, which should help achieve better clustering results. Fourth, different from Grad-CAM [59], it also works with convolutional DNN layers and regression DNNs.

LRP redistributes the relevance scores of neurons in a higher layer to scores of the lower layer. Assuming j and k to be two consecutive layers of the DNN, LRP propagates the relevance scores

---

[4]In a Convolutional Neural Network (CNN), a feature map (or activation map) is the matrix resulting from the application of a kernel to the input tensor [41].
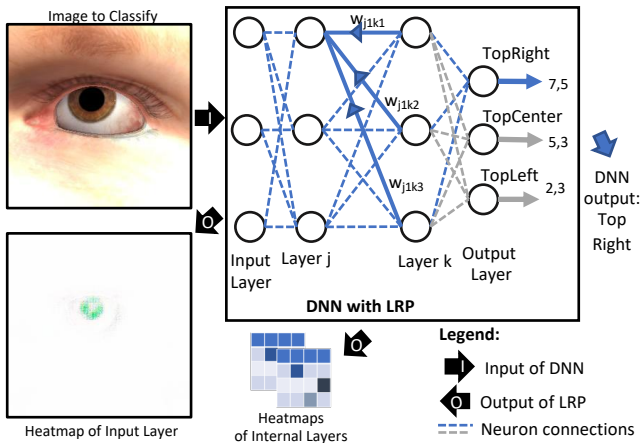

Fig. 1. Layer-Wise Relevance Propagation.

computed for a given layer k to a neuron of the lower layer j. It has been theoretically justified as a form of Taylor decomposition [48]. Fig. 1 illustrates the execution of LRP on a fully connected network used to classify inputs. LRP formulas for well-known layer types are available [47].

A heatmap is a matrix with entries in $\mathbb{R}$, i.e., it is a triple (N, M, f) where $N, M \in \mathbb{N}$ and f is a map $[N] \times [M] \to \mathbb{R}$. Hereafter, we use the syntax $H_{i,j}^L$ to refer to an entry in row i ($i < N$) and column j ($j < M$) of a heatmap H computed on layer L. The size of the heatmap matrix (number of entries) is $N \cdot M$, with N and M depending on the dimensions of the DNN layer L. For convolution layers, N captures the number of neurons in the feature map, while M captures the number of feature maps.

2.2 HUDD

HUDD [17] identifies the different situations in which an image-processing DNN is likely to trigger an erroneous output by generating clusters (i.e., root cause clusters, RCCs) containing misclassified input images. The images belonging to the same RCC share a common set of characteristics that are plausible causes for failures. HUDD is based on the intuition that since heatmaps capture the relevance of each neuron on DNN outputs, the processing of inputs sharing the same root cause (or hazard-triggering event) should lead to similar heatmaps. For this reason, to identify the root causes of DNN failures, HUDD relies on clustering based on heatmaps.

HUDD works as follows. First, for each failure-inducing image in the test set, it relies on LRP to generate heatmaps of internal DNN layers. Each entry of a heatmap for a specific DNN layer captures the relevance score of a neuron in that layer. Then, HUDD relies on a hierarchical agglomerative clustering algorithm [37] to group images. To measure the distance between two images, which is necessary for clustering, it computes the Euclidean distance between their heatmaps. To determine the number of clusters to generate, it relies on the knee-point method to identify the configuration in which the weighted intra-cluster distance (i.e., the distance between pair of images within a cluster, weighted by the relative size of the cluster) stops decreasing significantly.

Since certain input space regions might be less dense than others (i.e., include fewer data items), the weighted intra-cluster distance computed by HUDD balances cohesion and cluster size (i.e., it is acceptable to have a larger intra-cluster distance when the cluster has less elements). Also, since LRP can generate heatmaps for each DNN layer, HUDD generates a set of clusters for each DNN layer and selects the one that minimizes the weighted intra-cluster distance (i.e., the one with the most cohesive clusters).

HUDD has proven useful in identifying root causes (or hazard-triggering events) due to a diverse set of problems, including an incomplete training set, an incomplete definition of the predicted classes, and limitations in the simulator controls.

2.3 Multi-objective and Many-objective Optimization

Evolutionary methodologies are state-of-the-art solutions to find a set of nondominated solutions in multi-objective optimization problems. In software engineering, they are widely adopted to identify test inputs that maximize some coverage criterion [53]. Also, they have been successfully used to test autonomous driving systems [1, 5, 28, 49].

NSGA-II [1, 5, 12, 49] is a state-of-the-art solution for optimization problems with up to three objectives. It is shown in Fig. 2. NSGA-II works by first generating a random population (Line 2), which is evolved in a number of iterations (Lines 3 to 16). A new offspring (Q_t) is generated by applying mutation and crossover operators (Line 4). The algorithm preserves elitism since the new population (P_{t+1}) is generated by selecting the best individuals in the union of the current population (P_t) and the offspring (Line 5). The best individuals are selected by first ranking individuals based

Require: O = o1, o2, ... the objectives
Require: M, population size
Require: T, search budget (i.e., number of iterations to perform)
Ensure: a population whose Pareto front includes the best found solutions
1: $t \leftarrow 0$ //current generation
2: $P_t \leftarrow$ *CREATE RANDOM POPULATION WITH SIZE M*
3: **while** $t < T$ **do**
4: $\quad Q_t \leftarrow$ *GENERATE OFFSPRING FROM* P_t
5: $\quad R_t \leftarrow P_t \cup Q_t$
6: $\quad (F_1, F_2, ..., F_l) \leftarrow$ *APPLY* fast-nondominated-sorting *TO* R_t
7: $\quad r \leftarrow 0$ //counter for the Pareto front rank
8: $\quad$ **while** $|P_{t+1}| + |F_r| \le M$ **do** //iterate over all the Pareto fronts till they fit into the population
9: $\quad\quad P_{t+1} \leftarrow P_{t+1} \cup F_r$
10: $\quad\quad r \leftarrow r + 1$
11: $\quad$ *ASSIGN CROWDING DISTANCE TO INDIVIDUALS IN* F_r //see Fig. 3; F_r is the front that does not fully fit into P_{t+1}
12: $\quad$ *SORT* F_r *BASED ON CROWDING DISTANCE*
13: $\quad$ **while** $|P_{t+1}| \le M$ **do** //till P_{t+1} is not full, add the individuals in the sorted F_r
14: $\quad\quad ind \leftarrow$ *EXTRACT FIRST INDIVIDUAL IN* F_r
15: $\quad\quad P_{t+1} \leftarrow P_{t+1} \cup ind$
16: $\quad t \leftarrow t + 1$
17: **Return** P_t

Fig. 2. NSGA-II

Require: I, vector whose components represent an individual
1: $l = |I|$
2: **for** $i = 0; i < |I|$ **do**
3: $\quad I[i] = 0$
4: **for each** objective o **do**
5: $\quad I \leftarrow$ sort I according to objective o
6: $\quad I[1]_{distance} \leftarrow \infty$
7: $\quad I[l]_{distance} \leftarrow \infty$
8: $\quad$ **for** $i = 2$ to $(l-1)$ **do**
9: $\quad\quad I[i]_{distance} \leftarrow I[i]_{distance} + \frac{(I[i+1]_m - I[i-1]_m)}{(max\ fitness\ for\ o) - (min\ fitness\ for\ o)}$

Fig. 3. Crowding-distance-assignment function for NSGA-II. Red part indicates the instruction commented out in *NSGA-II'* to ensure that, for each objective, the best individual is preserved.

on their belonging to the l-th nondominated fronts (Line 6), which is efficiently performed by the fast-nondominated-sorting procedure [12]. Then, the algorithm adds individuals to the population according to their rank (Line 9), till it finds a ranked subset that cannot be fully added (Line 8). Finally, to preserve diversity in the population, NSGA-II selects the items belonging to the last ranked subset according to a crowding distance function that prioritizes individuals being more distant from others along with individuals at the boundaries of the objective space (Lines 11 to 15). Fig. 3 provides the pseudocode for the function that computes the crowding-distance for a set of individuals. Lines 6 and 7 prioritize the individuals with the best and worst fitness, respectively, by assigning them infinite distance.

The term many-objective is used for algorithms that tackle problems with more than three objectives [40]. A popular many-objective algorithm used in software testing is MOSA [53]. Given test cases (i.e., sequences of inputs for program APIs) as individuals, MOSA models branch coverage as a search objective but does not account for the length of individuals. Shorter test cases are, however, easier to read than longer ones and should be prioritized. Therefore, MOSA extends NSGA-II by relying on an archive that is used to keep track of the shortest test cases accidentally identified during the search. In addition, since MOSA aims to fulfil all the objectives and not only a subset of them, instead of relying only on the nondominance relation to select elitist individuals (i.e., Line 6 in Fig. 2), it makes use of the *preference criterion*, a strategy that ensures preserving individuals that minimize the objective score for uncovered objectives (i.e., objectives without an

individual already in the archive). To generate inputs for DNN-based systems with MOSA [25], it is necessary to specify a fitness threshold to determine objective coverage (e.g., a safety violation occurs if the ego vehicle gets too close to the vehicle in front). At every iteration, the archive is updated to keep the best individual found during the search; however, the probability of replacing an individual already in the archive is low. Indeed, thanks to the preference criterion, the population includes only the best individuals for the objectives not covered yet. When it is not possible to specify a threshold to determine when an objective is covered (e.g., our context), the main benefit of MOSA is to ensure that the best individual for each objective is preserved; indeed, the archive becomes useless because the best individuals are preserved directly in the population. However, a simple modification of NSGA-II can achieve the same purpose (see Section 3.1.2).

2.4 Diversity optimization

In our work, we aim to rely on evolutionary algorithms to generate a set of diverse images belonging to a RCC. As described in the related literature [26–28, 58], an image can be modelled as an individual (or chromosome) of the population, represented as a vector whose components capture the values assigned to simulator parameters used to generate the image.

Although, in principle, genetic algorithms can be used to evolve a population of individuals and achieve our objectives (i.e., generate individuals belonging to the RCC and maximize the diversity between them), the use of well-known algorithms for this purpose (e.g., NSGA-II) is not feasible because it is not possible to define a fitness function for diversity, as explained next.

To maximize diversity, since it is the property of a set of images, it is not possible to define a fitness function that assesses how much an offspring individual contributes to diversity prior to knowing what the final set will be. Also, note that selection based on crowding distance, which is part of NSGA-II, does not help because such distance is defined over the objective space, as opposed to the simulator parameter space. Consequently, the NSGA-II algorithm cannot simply be adopted to address our problem.

We can represent a solution for one of our objectives (i.e., a set of n diverse images belonging to a RCC) as an individual where each image is modelled with a subsequence of the chromosomes (e.g., the first subsequence of m chromosomes are for the first images, the second for the second image, and so on). This would enable *whole test suite generation* (i.e., the generation, within a same search run, of all the solutions for our objectives) [20]. However, different from traditional software testing, individuals achieving different objectives are unlikely to present dependencies. For example, in traditional software testing, nested conditions lead to dependencies between search objectives, whereas two RCCs should include images that are diverse (e.g., one picture of a person looking left and another picture with a person looking right); therefore, in our context, generating an image that fulfills one objective should not help with achieving another objective. Further, modelling multiple images as a single individual would lead to many simulator runs to generate a single individual, which leads to an inefficient exploration of the search space and, therefore, entails a large budget to generate the desired solutions. For these reasons we chose to model a single image as an individual.

An alternative solution consists of relying on an archive where, at each iteration of the search algorithm, we add the individuals in the first non-dominated front that contribute to increasing diversity. Related work suggests adding to the archive the individuals with a *sparseness* value (i.e., the distance from the nearest neighbour) that is above a set threshold (hereafter, *sparseness threshold*) [57, 58]. DeepJanus [58] and DeepMetis [57] are two state-of-the-art solutions that rely on a *sparseness threshold*. They extend the popular NSGA-II algorithm with an archive and with repopulation (to escape from stagnation by replacing the most dominated individuals with random ones); we refer to such algorithm as *DeepNSGA-II*. Unfortunately, in our context, the sparseness

threshold may directly affect the quality of the results; indeed, if the threshold is too low, most of the selected individuals will be similar to each other, thus making the learning algorithm derive rules that do not characterize the whole RCC. Similarly, if the threshold is too high, we end up selecting only a small number of individuals, which prevents the generation of accurate rules. Unfortunately, identifying an appropriate threshold requires multiple executions of the algorithm, which, in our context, shall be repeated for every RCC; indeed, since it may be easier for the simulator to generate certain images than others, some portions of the input space may be denser than others and, therefore, different thresholds might be needed for different RCCs. In practice, this would lead to an expensive process, which is practically infeasible. One simple solution is to rely on a sparseness threshold of zero (i.e., we add to the archive any image that differs from the ones already in it) [57]. However, such choice may render the algorithm less efficient as it leads to larger archives and the larger the archive the more costly the distance is to compute. Further, in our context (Section 3), the best individuals identified through the search (in the archive) are used as input for additional search iterations; therefore, the *DeepNSGA-II* algorithm should be combined with a strategy to select the best individuals in the archive (Section 4.2 provides details about the strategy adopted in our empirical evaluation).

In addition to the limitations indicated above, please note that DeepJanus and DeepMetis address problems that are different than ours. They both rely on two fitness functions, namely F_1 (to be maximized) and F_2 (to be minimized). In both DeepJanus and DeepMetis, at every search iteration, every individual of the population is added to the archive if F_1 is above the sparseness threshold and if F_2 is below a user-defined threshold capturing if the individual addresses the other objective of the search further described below. DeepJanus characterizes the frontier of DNN misbehaviours by identifying pairs of inputs that are close to each other, with one input leading to a correct DNN output and the other to a DNN failure [58]. Function F_1 focuses on sparseness and closeness of input pairs and is computed as follows:

$$F_1 = \{sparseness(i, A) - k * distance(i.m1, i.m2)\}$$

with $x.m1$ and $x.m2$ being a pair of inputs, *sparseness* the function that computes the distance from the nearest individual in the archive A, and *distance* the distance between two individuals. Function F_2 focuses on the closeness to the frontier and is computed as

$$F_2(i) = \begin{cases} eval(i.m1) \cdot eval(i.m2) & \text{if} > 0 \\ -1 & \text{otherwise} \end{cases}$$

with *eval being the difference between (a) the confidence level associated with the expected label and (b) the maximum confidence level associated to any other class*. Based on the above, DeepJanus cannot be directly applied in our context because we do not aim to generate pairs of individuals. DeepMetis aims to identify inputs leading to DNN failures in mutated DNNs (i.e., DNN models trained or modified to be less accurate than the DNN under test) [57]. Different from DeepJanus, in DeepMetis, an individual is a single image. Function F_1 focuses only on sparseness

$$F_1 = \{sparseness(i, A)\}$$

Function F_2 measures how well an individual triggers inaccurate outputs in mutated DNNs and is computed as

$$F_2 = \sum_{mutant \in Mutant} eval_{mutant}(i)$$

with $eval_{mutant}$ returning the *difference between the confidence associated with the expected class and the maximum confidence associated with any other class when the prediction is correct (-1 if the prediction is wrong)*. Although F_1 might also be used in our context to drive the generation of sparse

solutions, F_2 needs to be modified to capture our goals (i.e., the input belongs to a *RCC* and leads to a failure).

To address our problem, we thus defined an algorithm (i.e., the PaiR algorithm, described in Section 3.1) that takes advantage of every image generated by the simulator in an iteration and does not require a sparseness threshold. Further, we have separated the identification of images belonging to a RCC, which is achieved by PaiR, from the identification of failing images, which is achieved with additional search iterations performed through an extension of the NSGA-II algorithm. In Section 4.2, we compare PaiR with a DeepNSGA-II solution with a sparseness threshold of zero and F_2 matching the same fitness function used by PaiR to identify individuals belonging to a *RCC*. PaiR does not support whole test suite generation but it is executed once for each RCC.

2.5 Rule learning algorithms

In our work, we aim to derive properties that characterize unsafe images. Given a dataset capturing properties of safe and unsafe images (e.g., the parameter values used to generate an image with a simulator), our objective can be achieved by relying on machine learning algorithms that derive *decision rules* [46].

A decision rule is an IF-THEN statement consisting of a condition and a prediction. For example, in our context, a decision rule may indicate that if the horizontal angle used to generate an image is above 50 degrees (i.e., the head of the person is so turned that her left eye is barely visible), then the image is failure-inducing. Decision rules are machine learning models that can be easily interpreted by humans [46], which motivates our choice.

The support of a rule indicates the percentage of instances to which the condition of a rule applies, while its accuracy measures the proportion of classes correctly predicted for the instances to which the condition of the rule applies.

To combine multiple rules, rule learning algorithms derive either decision lists or decision sets. A decision list is ordered, and we should rely on the prediction of the first rule whose condition evaluates to true. In a decision set, predictions are based on some strategy (e.g., majority voting). Since we aim to rely on the generated rules to derive logical expressions, for our work, we selected algorithms deriving decision lists which can be directly translated into a single expression (see Section 3.2). In particular we rely on the PART algorithm [19], which combines the approach of RIPPER [9], a rule learning algorithm, with decision trees, which have been successfully applied in related work to characterize DNN inputs [26].

PART, similarly to RIPPER, relies on a *separate-and-conquer* approach. It relies on a partial C4.5 decision tree to derive each rule. It starts by deriving a rule that provides accurate predictions for some of the data points. Such result is achieved by building a pruned C4.5 decision tree from all the data points, and then selecting the leaf with the largest support to transform it into a rule. The decision tree is then discarded and all the data points that are covered by the rule are excluded from the training set. This rule learning procedure is repeated until the whole data set is processed.

Since, in our context, for each image under analysis we know (1) the DNN outcome (i.e., a DNN failure or a correct output) and (2) the set of simulator parameter values used to generate it, we can configure PART to consider the DNN outcome as the class to predict and the simulator parameter values as the features to derive rules from.

Fig. 4 provides an example output generated by PART when processing images from our case study subjects. Each image depicts a person's head and is represented by a number of simulator parameters configured to generate the image (an example is shown in Fig. 5). The considered parameters include, among others, the orientation of the head (pitch, yaw, and roll captured by the parameters $HeadPose_X$, $HeadPose_Y$, and $HeadPose_Z$).

1: $HeadPose_Y > 50.34 : class = DNN - error$
2: $HeadPose_Y < 13.34 : class = DNN - correct$
3: $HeadPose_Z > 60 \& HeadPose_Y > 30 : class = DNN - error$
4: $HeadPose_Z \leq 60 : class = DNN - correct$
5: (default) : $class = DNN - error$

Fig. 4. Example PART output

Fig. 5. Example image generated with IEE-Humans, one of the simulators used in our empirical evaluation. It is labeled as looking top-right (i.e., towards the top-right corner of the picture).

In Fig. 4, Line 1 indicates that we observe a DNN failure if the value of the parameter $HeadPose_Y$ is above 50.34 (i.e., the person is looking to the right of the viewer, with the head turned so much that her left eye is barely visible). Line 2 indicates that, when the rule in Line 1 does not apply and the person is looking center or to her right (i.e., if $HeadPose_Y$ is below 13.34), then the DNN produces a correct output. Line 3 indicates that, when the two rules above do not hold, we observe a DNN failure if the value of parameter $HeadPose_Z$ is above 60 and the value of parameter $HeadPose_Y$ is above 30 (i.e., when the head is tilted and the person is looking to her left then the DNN makes a mistake even if her left eye is visible). Line 4 indicates that, when the three rules above do not hold, the DNN output is correct if the value of parameter $HeadPose_Z$ is below 60 (i.e., the head is not tilted). Finally, Line 5 is the default rule, which indicates that, when all the rules above do not hold, the DNN output is incorrect.

3   SIMULATOR-BASED EXPLANATIONS FOR DNN FAILURES

In this paper, we propose *Simulator-based Explanations for DNN failurEs (SEDE)*, an approach to characterize the root causes (events) leading to DNN failures; we call such events hazard-triggering events. It targets contexts in which DNNs are partly trained using simulators, which is common practice in safety-critical contexts with complex inputs [6, 31, 36]; for example, DNNs implementing vision-based driving tasks or interpreting human postures. This is the case for our industry partner, which relies on a simulator capable of generating images of human bodies, seated in a car environment, to train DNNs that interpret human postures (e.g., determine gaze or drowsiness).

The reference DNN training process is depicted in Fig. 6. It leverages simulators to reduce the cost related to collecting and labelling a large number of images. In general, engineers tend to initially train the DNN using images automatically generated with a high-fidelity simulator (e.g., a human body simulator). When the DNN is deemed accurate, to enable the correct processing of real-world images, engineers fine-tune the trained DNN using real-world images (e.g., images of real people seated in a car). Fine-tuning consists of relying on the DNN training algorithm to update

the weights of the DNN trained with simulator images; retraining may concern the whole set or a subset of the DNN weights (e.g., the fully connected layers in a CNN). The fine-tuned DNN is then tested with real-world images; in current practice, engineers visually inspect the images leading to DNN failures and determine what are the hazard-triggering events. SEDE, instead, automatically characterizes the hazard-triggering events observed in a test set with real-world images; such objective is achieved by generating an expression that constrains the parameters of the simulator used to generate the training set.

SEDE works in four steps depicted in Fig. 7. In the *first step*, SEDE relies on a state-of-the-art solution (HUDD, see Section 2.2), to automatically identify clusters of images (root cause clusters, RCCs) leading to a DNN failure because of common hazard-triggering events.
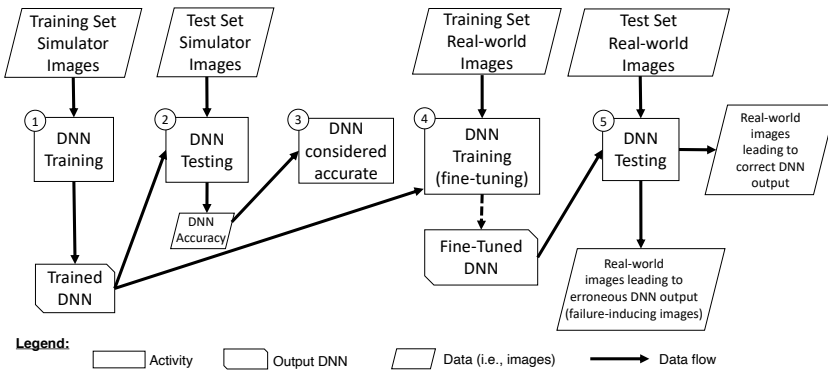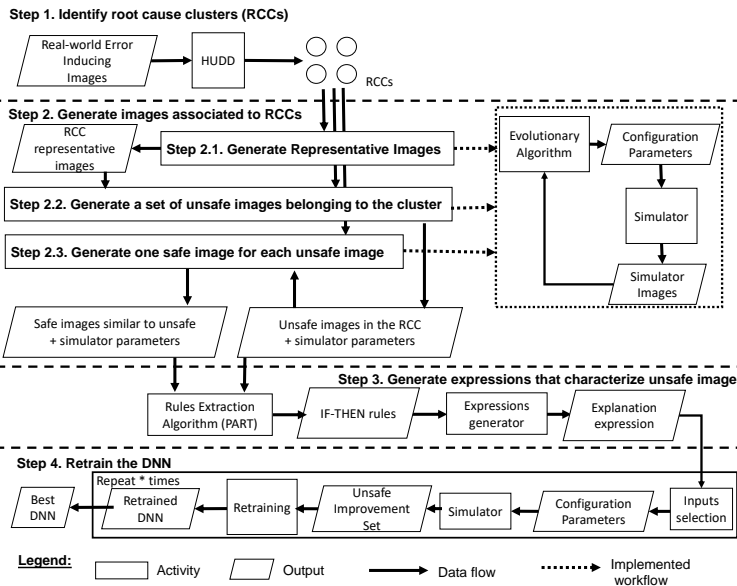

Fig. 6. DNN training and testing

Fig. 7. The SEDE workflow

In the *second step*, SEDE relies on evolutionary search driven by the analysis of heatmaps and simulator parameters to generate images that are associated to RCCs, enabling accurate learning to characterize unsafe scenarios. To precisely derive constraints on simulator parameter values (configurations) for distinct hazard-triggering events, while avoiding overfitting, for each RCC, we are interested in generating many diverse images leading to correct and erroneous DNN outputs. Addressing multiple hazard-triggering events entails multiple search runs, as justified in Section 3.1. More precisely, for each RCC, our algorithm executes one search to generate a set of diverse images belonging to that RCC, one search to generate a set of unsafe (i.e., failure-inducing) images that are close to each of the diverse images and belonging to the RCC, and another search to identify a set of safe images (i.e., not failure-inducing) that are close to the unsafe ones. For classifier DNNs, a failure is triggered when the output class differs from the ground truth, where the latter is automatically derived from the parameters of the simulator. In the case of regression DNNs, a DNN failure is triggered when the difference between the DNN output and the reference value (ground truth) is above a set threshold, which is defined by engineers based on domain knowledge.

In the *third step*, SEDE relies on the availability of a sufficiently large number of diverse safe and unsafe images, generated in the previous step, to identify those parameter ranges that characterize unsafe images. More precisely, SEDE relies on a machine-learning algorithm (i.e., PART) to extract such ranges, under the form of logical expressions that accurately predict and therefore represent unsafe images.

In the *fourth step*, SEDE relies on the derived expressions to generate an additional set of images that are used to retrain and improve the DNN.

Please note that SEDE can also be applied to test sets generated using simulators; however, compared to test sets with real-world images, simulator-based failure-inducing test sets are less challenging to characterize. Indeed, when test set images are generated using simulators, decision trees or any other rule learning algorithm can be used to identify the commonalities among the failure-inducing images [2, 27]. For example, it is sufficient to generate RCCs with HUDD and then extract PART rules from them. However, if the number of available failure-inducing images is limited, learning may not lead to accurate results. In such cases, SEDE may be useful to generate additional failure-inducing images, thus enabling the generation of more accurate explanatory rules.

Below, we describe Steps 2 to 4 since Step 1 simply involves the execution of HUDD to generate RCCs.

3.1   Step 2 - Automated Generation of Unsafe and Safe Images

In the second step of SEDE, for each RCC, we aim to automatically generate images that (goal 1) exhibit the same hazard-triggering events observed in the RCC and (goal 2) enable a learning algorithm to extract accurate rules predicting unsafe images. These rules shall ideally characterize parameter values that are observed only with unsafe images belonging to a specific RCC.

To achieve goal 1, it is sufficient to explore the input space by generating diverse images that are similar to the ones belonging to the RCC.

To achieve goal 2, instead, we need to generate both safe images and unsafe images. To be accurate, the rules derived by SEDE shall provide ranges for parameter values that are observed only or mostly with unsafe images and are as wide as possible. To this end, the unsafe images used to generate the rules must be as diverse as possible and for each unsafe image, we should identify a very similar safe image so that the algorithm learns to precisely distinguish unsafe images from safe ones, i.e., precisely learn the boundary between them.

To achieve the above, we rely on a divide-and-conquer approach that consists in executing three genetic algorithms for each RCC in a sequence. We make use of genetic algorithms because they have been successfully used by related work to explore the input space of DNNs using simulators [58].

First (Step 2.1), we generate diverse images belonging to the RCC, safe or unsafe. The goal is to cover the cluster with representative images.

Second (Step 2.2), we rely on the generated representative images to generate an additional set of unsafe and diverse images belonging to the cluster; this is achieved by generating, for each representative image, one unsafe image that is close to it and belongs to the RCC.

Third (Step 2.3), for each unsafe image derived by the second evolutionary search, we generate one safe image that is as close to it as possible.

In summary, we aim to generate a sufficient number of diverse safe and unsafe images belonging to each RCC to enable effective learning.

3.1.1 Step 2.1 - Generate Representative Images.

We have two objectives: (1) generate a set of images that belong to a cluster and (2) minimize their similarity (i.e., maximize their diversity); however, since RCCs tend to contain images that are similar, the two objectives are unlikely to compete. Indeed, two dissimilar images are unlikely to belong to the same cluster. Further, for our purpose, it is useless to generate a set of diverse images if they do not belong to the RCC. Consequently, it is not desirable to rely on a multi-objective search algorithm to address them. To drive the search process, we thus need a fitness function that enables an evolutionary search algorithm to first generate images that belong to a cluster and then decrease their similarity.

In Section 2.4 we clarified why state-of-the-art approaches are inapplicable or ineffective when applied to achieve our objectives; in this section, we propose a dedicated fitness function and algorithm.

In the presence of a function that measures the distance of an individual i from the center of the RCC C (hereafter, $RCC_{distance}(C, i)$) and a function that measures how an individual contributes to minimizing the similarity across cluster members (hereafter, $F_{similarity(i)}$), to obtain a mathematically adequate behavior, our fitness function $F_1^C(i)$ can be defined as follows:

$$F_1^C(i) = \begin{cases} F_{similarity}(i) & \text{if } RCC_{distance}(C, i) \leq 1 \\ RCC_{distance}(C, i) & \text{otherwise} \end{cases} \tag{1}$$

Our goal is to minimize $F_1^C(i)$ and given that (1) $F_{similarity(i)} \leq 1$ (see Equation 7 below) and (2) $RCC_{distance} \leq 1$ is only true when the image belongs to the RCC, we obtain the targeted property: fitness is always lower in cases where an individual i belongs to the RCC. Below, we describe the functions $F_{similarity}$ and $RCC_{distance}(C, i)$.

To determine if an individual belongs to a RCC, we can measure its distance from the RCC centroid and verify if it is shorter than the RCC radius (i.e., the max distance between the centroid and the farthest image). For the distance metric, as in HUDD, we can use the Euclidean distance between two heatmaps (hereafter, *HeatmapDistance*); however, to generate a heatmap we need a concrete image and, therefore, instead of relying on the cluster centroid, we identify its medoid. The formula for the *HeatmapDistance* function is:

$$HeatmapDistance(A, B) = \sqrt{\sum_{m=1}^{M} \sum_{n=1}^{N} (A_{m,n}^L - B_{m,n}^L)^2} \tag{2}$$

where A and B are two heatmap matrices generated with LRP, and m, n indicate the row and column of a cell in a matrix (M and N indicate the total number of rows and columns, respectively). Since LRP generates one heatmap matrix for every neuron layer of the DNN, we rely on the heatmap generated for layer L as selected by HUDD to generate the RCC.

Similar to related work [25, 26], within our evolutionary algorithms, we represent an individual using a vector of simulator parameter values. For this reason, to compute the heatmap distance, our search algorithm, for every offspring individual, first generates one image using the simulator, then processes the generated image using the DNN under test, and finally generates its heatmap using the LRP algorithm.

The medoid of a RCC C is the image that minimizes the average pairwise distance from the other images of the RCC; it is computed as follows:

$$Medoid(C) = \arg\min_{y \in C} \left\{ \frac{\sum\limits_{x \in C, x \neq y} HeatmapDistance(x, y)}{|C| \cdot (|C| - 1)} \right\} \qquad (3)$$

The radius of a RCC C is the maximum distance between its medoid and any other image in C:

$$Radius(C) = \arg\max_{y \in C} \left\{ HeatmapDistance(y, Medoid(C)) \right\} \qquad (4)$$

Since clusters may have different radiuses, we compute $RCC_{distance}(C, i)$ for a RCC C as the heatmap distance between individual i and C's medoid $Medoid(C)$, normalized by C's radius $Radius(C)$:

$$RCC_{distance}(C, i) = \frac{HeatmapDistance(i, Medoid(C))}{Radius(C)} \qquad (5)$$

To compute $F_{similarity}(i)$, we follow related work [58] and measure how much an individual contributes to the diversity of the population by measuring its distance from the closest individual in the population.

We can measure the distance between two individuals i and j based on their chromosome vectors v_i and v_j (containing simulator parameter values), as follows:

$$
\begin{aligned}
ChromosomeDistance(i, j) &= \cos(i, j) \qquad (6)\\
&= \frac{v_i \cdot v_j}{\|v_i\| \|v_j\|}\\
&= \frac{\sum\limits_{p=1}^{|i,j|} (v_i[p] \cdot v_j[p])}{\sqrt{\sum\limits_{p=1}^{|v_i|} v_i[p]^2} \sqrt{\sum\limits_{p=1}^{|v_j|} v_j[p]^2}}
\end{aligned}
$$

where $\cos(i, j)$ is the cosine similarity between v_i and v_j, whose range is $[0, 1]$; $v_i[p]$ and $v_i[p]$ indicate the value of the p-th component in the vectors v_i and v_j, respectively.

F_1^C can be used to drive the search algorithm, described below.

The SEDE search algorithm: PaiR. Pairwise Replacement (PaiR) is our algorithm to generate representative RCC images; it is presented in Fig. 8 and described below.

PaiR aims to evolve a whole population of individuals to generate a population of individuals that both belong to the RCC under analysis and are diverse. The size of the population is a parameter of the algorithm. PaiR leverages all the images generated by the simulator at every search iteration by

Require: RCC_C, the RCC under analysis
Require: s, population size
Require: r, number of random populations of images to generate initially
Require: b, search budget (i.e., number of iterations to perform)
Ensure: P_1^C, an optimal set of diverse images belonging to cluster RCC_C
1: **repeat**
2:     $R \leftarrow R \cup \{RANDOM\ POPULATION\ with\ size\ s\}$
3: **until** r random populations have been generated
4: $t \leftarrow 0$
5: $P \leftarrow$ selects the population in R with the best fitness
6: **while** $t < b$ **do**
7:     EVALUATE the FITNESS of P based on RCC_i
8:     $O \leftarrow$ GENERATE the OFFSPRING population from P
9:     EVALUATE the FITNESS of O based on P and RCC_C
10:     **while** $SIZE(O) > 0$ **do**
11:         SORT O based on decreasing FITNESS
12:         $i_o \leftarrow$ extract and remove first individual in O
13:         **if** there is at least one individual in P that is outside the RCC **then**
14:             $i_p \leftarrow$ IDENTIFY the individual in P with the highest fitness
15:         **else**
16:             $i_p \leftarrow$ IDENTIFY the individual in P that is closer to i_o
17:         **if** fitness of i_o < fitness of i_p **then**
18:             Within P, replace i_p with i_o
19:             EVALUATE the FITNESS of P based on RCC_C
20:             EVALUATE the FITNESS of O based on P and RCC_C
21:     $t \leftarrow t + 1$
22: $P_1^j \leftarrow$ all the individuals in P that belong to RCC_C
23: **return** P_1^C
24: **exit**

Fig. 8. PaiR, the Genetic Algorithm integrated into SEDE

replacing one or more images in the parent population with offspring individuals having a better fitness.

PaiR works by looking for individuals that minimize F_1^C. At every iteration, PaiR creates a new population P' obtained by replacing an individual i_p with an individual i_o having a better fitness (i.e., $F_1^C(i_o) < F_1^C(i_p)$), as described in the following.

The PaiR algorithm receives as input the RCC under analysis (hereafter, RCC_C), the configuration parameters s (population size) and r (number of random populations to generate initially), and the search budget b indicating the number of iterations to perform.

To maximize the chances of finding an optimal solution, as in related work [64], the PaiR algorithm starts by generating several random populations (Line 1 to 3) and selecting the population including the individual with the lowest fitness value (P, Line 5).

In its main loop (Lines 6 to 21), PaiR computes the fitness values of individuals in the parent population P and, from P, generates an offspring population O by relying on the same strategy commonly adopted with NSGA-II in similar contexts (i.e., tournament selection, simulated binary crossover, crossover probability p_c, and mutation probability p_m).

The offspring individuals are sorted to process first the ones with better fitness (Line 11). PaiR considers each individual in O (i_o) as a replacement of an individual in P (i_p), if the former has a better fitness. PaiR distinguishes between the cases in which P only contains individuals belonging to the RCC (Line 15) and when at least one individual does not (Line 13). Indeed, if we do not have enough individuals belonging to the RCC, it is not important to maximize their diversity.

If P contains at least one individual not belonging to the RCC, PaiR simply looks for the individual in P with the highest fitness (i.e., the one that is furthest from the cluster medoid). Such individual will be replaced by i_o if the latter has a lower fitness (Line 17), which is always the case if i_o belongs

to the RCC; otherwise, it depends on the value of $RCC_{distance}$: i_o replaces i_p if i_o is closer to the RCC medoid.

If all the individuals in P belong to the RCC, PaiR selects the individual in P that is closer to i_o (i.e., i_p, Line 16). The offspring individual i_o replaces i_p if i_o has a lower fitness than i_p (Line 17). Since i_p belongs to the RCC, according to Equation 1, i_o has a lower fitness than i_p when i_o belongs to the RCC and has a lower $F_{similarity}$.

$F_{similarity}(i)$ is computed differently for individuals in the parent and in the offspring population; the reason is that we must decide if an individual i_o in the offspring population O, would lead to a population P' with a larger diversity than P. P' is obtained by replacing an individual i_p in the parent population P with i_o; therefore, we obtain a different P' from a same P, depending on the selected individuals i_p and i_o. To help identify the most diverse P' and avoid computing the similarity for all possible P' obtained from every individual in O, inspired by related work [58], we compare (a) the distance between i_p and its closest neighbour in P (i.e., the closest individual in P excluding i_p itself) with (b) the distance between i_o and the closest individual in P'. Note that P' includes i_o and $P \setminus \{i_p\}$. Therefore, for an individual i_p in the parent population P, we compute one minus its distance from $closest_{i_p}$, the closest neighbour to i_p

$$F_{similarity(i_p)} = 1 - ChromosomeDistance(i_p, closest_{i_p}) \qquad (7)$$

For an individual i_o in the offspring population O, we compute one minus its distance from $closest_{i_o}$, the individual in $P \setminus \{i_p\}$ that is closest to i_o:

$$F_{similarity(i_o)} = 1 - ChromosomeDistance(i_o, closest_{i_o}) \qquad (8)$$

Based on the above, given two individuals i_o and i_p in the offspring and parent populations, respectively, the individual i_o has a better fitness than i_p if they both belong to the RCC and the similarity between i_o and the closest individual in $P \setminus \{i_p\}$ is lower than the similarity between i_p and its closest individual in P. Since the distance from the closest individual should increase (lower similarity) when replacing i_p with i_o in the population, we can assume that a population P' will have higher diversity than P, which we demonstrate in Section 4.2.

The process is repeated until O is empty (Lines 10 to 20); since F_1^C depends on the distance between individuals being close to each other, given that both O and P vary over iterations, the fitness of the two populations is recomputed after every replacement (Lines 19 and 20).

After b iterations, the algorithm has generated a population P_1^C of individuals that are diverse and very likely to belong to RCC_C.

3.1.2 Step 2.2 - Generate a set of unsafe images belonging to the cluster.

For each RCC_C, we aim to generate a set of diverse, unsafe images, belonging to the cluster. To preserve the diversity of the images generated by PaiR in Step 2.1 (P_1^C), we can identify, for each image in P_1^C, an image that is close to it and makes the DNN fail. We can thus model our problem as a many-objective optimization problem with q objectives, where q is the number of images in P_1^C.

We define q fitness functions, one for each individual in P_1^C. An individual is an image generated with the simulator, modelled as described in Section 3.1.1. All fitness functions share the same

parameterized formula $F_2^{C,t}$, where $t \in \{1, ..., q\}$:

$$F_2^{C,t}(i) = \begin{cases} ChromosomeDistance(i, P_1^C[t]) & if\ (RCC_{distance}(RCC_C, i) \leq 1) \wedge (CLOSEST(i, P_1^C) = P_1^C[t]) \\ & \wedge (DNN_{failure}(i)) \\ 1 + F_{uncertainty}(i) & if\ (RCC_{distance}(RCC_C, i) \leq 1) \wedge (CLOSEST(i, P_1^C) = P_1^C[t]) \\ & \wedge (DNN_{correct}(i)) \\ 2 + ChromosomeDistance(i, P_1^C[t]) & if\ (RCC_{distance}(RCC_C, i) \leq 1) \wedge (CLOSEST(i, P_1^C) \neq P_1^C[t]) \\ 3 + RCC_{distance}(RCC_C, i) & if\ RCC_{distance}(RCC_C, i) > 1 \end{cases}$$

$$(9)$$

$F_2^{C,t}$ is a piecewise-defined function implemented through four sub-functions that return a value in the range $[0, 1]$ incremented by a constant (from 0 for the first sub-function to 3 for the fourth sub-function).

The fourth sub-function of the equation drives the search towards generating an individual belonging to RCC_C; indeed, when this is not the case, $F_2^{C,t}$ measures how far the individual is from the RCC medoid with $RCC_{distance}(RCC_C, i)$. Since not belonging to RCC_C is the worst case for an individual, $F_2^{C,t}$ must be higher than in other cases. This is achieved by returning $3 + RCC_{distance}(RCC_C, i)$, given that the codomain of the first three sub-functions referred to in $F_2^{C,t}$ lay in the range $[0, 3]$.

The third sub-function of the equation ensures that we generate one image for each individual in P_1^C; indeed, if the individual i belongs to RCC_C but the individual in P_1^C that is closest to i is not the t-th individual (i.e., $CLOSEST(i, P_1^C) \neq P_1^C[t]$), $F_2^{C,t}$ returns the cosine distance between i and the t-th individual in P_1^C, incremented by 2.

The second sub-function helps generating an individual that makes the DNN fail. Indeed, if the individual i belongs to RCC_C, is the closest to the t-th individual targeted by $F_2^{C,t}$, but does not make the DNN fail, then the fitness function returns a measure of DNN uncertainty increased by one.

To compute the uncertainty component of the fitness, we return one minus the cross-entropy loss function, which is commonly used to measure uncertainty:

$$F_{uncertainty}(i) = 1 - entropy(i) \qquad (10)$$

Cross-entropy measures how uncertain is the DNN about the provided output where lower cross-entropy values indicate higher certainty; therefore, by using one minus cross-entropy loss, we direct the search (minimization) towards the identification of images for which the DNN is less certain about outputs and, therefore, likely to produce an erroneous output.

The first sub-function of $F_2^{C,t}$, instead, for individuals that make the DNN fail, helps select the individual that is closest to the individual targeted by $F_2^{C,t}$; indeed, it returns the chromosome distance between the individual i and the individual $P_1^C[t]$.

Search Algorithm. To address our problem, we rely on a modified version of NSGA-II. NSGA-II is known to underperform in the presence of a large number of objectives (>3), mainly because of the exponential growth in the number of non-dominated solutions required for approximating the Pareto front [33]. However, we do not aim to find, as a solution to our problem, one single individual (i.e., image) that finds the best balance among different objectives but a set of images, each optimizing one independent objective (indeed, each image shall be close to a reference image in P_1^C); therefore, we are unlikely to find a set of nondominated solutions larger than $|P_1^C|$ (i.e., we will find one solution for each objective). Also, by definition, $F_2^{C,t}$ is always different than zero, which renders ineffective algorithms that look for an individual that *covers* an objective like MOSA. A solution to enable the adoption of MOSA would be to set a threshold to determine when the

objective has been *covered*; however, since RCCs differ with respect to their radius, we choose not to introduce a threshold that may lead to varying performance results across RCCs. Since we aim to optimize all the objectives, we rely on an extended version of NSGA-II with a modified crowding-distance function that ensures we select the minimal value for each objective, thus resembling the preference criterion of MOSA. Finally, we do not require an archive because the population will always include the best individual found for each objective and we do not need to evaluate an individual according to objectives not explicitly modeled (e.g., inputs' length in MOSA).

We propose a modified version of NSGA-II (hereafter, *NSGA-II'*) that includes a modified crowding-distance-assignment which ensures preserving, for each objective, the individual with the lowest fitness value. Our modified crowding-distance-assignment is shown in Fig. 3 (Page 6); different from the original crowding-distance-assignment used by NSGA-II, for each objective, we assign infinite crowding distance only to the individual that minimizes the fitness for the objective. NSGA-II, instead, assigns infinite distance also to the individual that maximizes the fitness for the objective (Line 7 in Fig. 3). As in NSGA-II, if more than one individual has the same minimal fitness, *NSGA-II'* assigns infinite distance only to one, randomly selected individual. Our choice ensures that, when the Pareto front includes a number of individuals larger than the population size s, for each objective, we preserve the individual that minimizes the fitness value. When the Pareto front has a number of individuals lower than the population size s, our crowding-distance assignment ensures the selection of individuals that minimize the fitness and are likely unsafe, which is what we intend to preserve in the final population. However, this situation is unlikely. Indeed, it may occur only when one individual has the same fitness value for several (i.e., z, with $z \leq q$) objectives, in one of the following unlikely scenarios:

- one image has the lowest $RCC_{distance}$ and no other individual falls in the cases covered by the first three sub-functions in $F_2^{C,t}$, for the same z objectives;
- one image has the lowest $ChromosomeDistance(i, P_1^C[t])$ for z reference images and no other individual falls in the cases covered by the first two sub-functions in $F_2^{C,t}$, for the same z objectives;
- one image has the lowest $F_{uncertainty}(i)$ and no other individual falls in the cases covered by the first sub-function in $F_2^{C,t}$, for the same z objectives;
- one image is failure-inducing and has the lowest $ChromosomeDistance(i, P_1^C[t])$, for the same z objectives.

We apply *NSGA-II'* for k iterations. Please note that when P_1^C already includes unsafe images belonging to the RCC C, *NSGA-II'* simply retains such images at every iteration; indeed, their fitness is already optimal according to $F_2^{C,t}$. After the k-th iteration of *NSGA-II'*, SEDE generates a population of individuals that likely lead to a DNN failure, which we refer to as P_2^C; at the end of the search, images not leading to DNN failures are removed from P_2^C.

3.1.3 Step 2.3 - Generate one safe image for each unsafe image.

We aim to generate a set of images that are similar to the unsafe images in P_2^C but do not lead to a DNN failure. As for the previous case, we model our problem as a many-objective optimization problem with q objectives, where q is the number of images in P_2^C; for each image in P_2^C, we aim to generate an image that is close to it and makes the DNN pass. We rely on the same *NSGA-II'* configuration used for Step 2.2 except for the fitness function, which in this step needs to be adapted to drive the generation of safe images; also, it is not necessary for these generated images to belong to RCC_C (indeed, we may not have any safe image within RCC_C). In Step 2.3, *NSGA-II'* evolves a population P_3^C that initially matches P_2^C.

As for Step 2.2, we define q fitness functions, one for each image in P_2^C; these functions share the same parameterized formula, for $t \in \{1, ..., q\}$:

$$F_3^{C,t}(i) = \begin{cases} ChromosomeDistance(i, P_2^C[t]) & \text{if } (CLOSEST(i, P_2^C) = P_2^C[t]) \wedge (DNN_{correct}(i)) \\ 1 + entropy(i) + |1 - RCC_{distance}(RCC_C, i)| & \text{if } (CLOSEST(i, P_2^C) = P_2^C[t]) \wedge (DNN_{failure}(i)) \\ 2 + ChromosomeDistance(i, P_2^C[t]) & \text{if } CLOSEST(i, P_2^j) \neq P_2^C[t] \end{cases}$$

(11)

The third sub-formula in $F_3^{C,t}$ matches the third formula in $F_2^{C,t}$ and aims to generate one image close to each each unsafe image in P_2.

The definition of the second sub-formula in $F_3^{C,t}$ is motivated by the fact that, since the initial population is P_2^C, all the images in the population initially cause a DNN failure and, therefore, the fitness should drive the search algorithm to find a close image leading to a correct output. To do so, we should look for images that either increase the confidence of the DNN output (i.e., leading to a lower entropy) or are close to the border of the RCC. Concerning the latter, since a RCC characterizes unsafe images, we expect that images next to its border are similar to the ones in the RCC but are less likely to be failure-inducing. In addition, moving further away from the border is expected to lead to images that are increasingly different from the images in the RCC. Such observations are captured by the absolute difference between 1 and $RCC_{distance}$ (i.e., how close the image is to the RCC border), which is added to $entropy(i)$ to help generate safe images.

The codomains of the second and the third sub-formula overlap as the former may return a fitness value above 3; this choice reflects the fact that an image being far away from the RCC border (when $|1 - RCC_{distance}(C, i)| > 1$) is unlikely to provide useful information as it is not helpful to derive rules that characterize safe images that are not similar to the ones in the RCC. Therefore, such image would be as useless as an image that is not close to the target image (i.e., $P_2^C[t]$). Also, if the DNN is highly uncertain about an image i (i.e., $entropy(i)$ is close to 1), it is unlikely for that image to help driving the algorithm towards a correct output. For the reasons above, the output of the second sub-formula falls into the codomain of the third sub-formula when the sum $entropy(i) + |1 - RCC_{distance}(RCC_C, i)| > 2$.

The first sub-formula in $F_3^{C,t}$, in the presence of images leading to correct DNN outputs, aims to minimize the distance from the t-th image; therefore, it returns the chromosome distance between the individual i and the target image $P_2^C[t]$.

The algorithm $NSGA\text{-}II'$ terminates after k iterations with a population P_3^C including images that are likely safe and close to the images in P_2^C; though unlikely, images leading to DNN failures are removed from P_3^C before SEDE's Step 3.

3.2 Step 3. Characterize unsafe images

In Step 3, we aim to generate an expression that characterizes unsafe images by using PART, to learn decision rules. For example, we could learn that images likely lead to a DNN failure when the parameters $HeadPose_X > 10$ and $HeadPose_Y > 50$.

PART generates as output a list of mutually exclusive rules (see Section 2.5) that predict the class of a data point; in our context, these rules predict the DNN output generated for an image (i.e., wrong or correct output) based on the values of the simulator parameters used to generate the image.

Since in Step 2, for each RCC, SEDE generated a set of unsafe (P_2^C) and safe (P_3^C) images, associated with simulator parameters, to learn decision rules with PART, SEDE selects from P_2^C all the images that belong to the RCC and are failing, and all the images from P_3^C that are not failing.

To generate an expression from the PART output (hereafter, referred to as *PART expression*), we implemented a procedure that, for all the rules leading to a DNN failure, generates a subexpression that joins the negation of all the preceding rules with the current rule. The generated subexpressions are mutually exclusive. For the example in Fig. 4, it leads to the following expression:

$$
\begin{aligned}
&(HeadPose_Y > 50.34) \parallel \\
&(\neg(HeadPose_Y > 50.34) \& \neg(HeadPose_Y < 13.34) \\
&\quad \&(HeadPose_Z > 60 \& HeadPose_Y > 30)) \parallel \\
&(\neg(HeadPose_Y > 50.34) \& \neg(HeadPose_Y < 13.34) \\
&\quad \& \neg(HeadPose_Z > 60 \& HeadPose_Y > 30) \\
&\quad \& \neg(HeadPose_Z <= 60))
\end{aligned}
\tag{12}
$$

Indeed, for Line 1 in Fig. 4, SEDE simply reports the expression generated by PART (there are no preceding expressions), which leads to "$HeadPose_Y > 50.34$". For Line 2, SEDE does not generate any subexpression because it concerns cases in which the DNN generates a correct output; the same happens for Line 4. For Line 3, SEDE negates the preceding expressions (i.e., "$HeadPose_Y > 50.34$" and "$HeadPose_Y < 13.34$") and joins them with the PART expression of Line 3 (i.e., "$HeadPose_Z > 60 \& HeadPose_Y > 30$"). For Line 5, SEDE simply generates a subexpression joining the negation of all the preceding expressions.

Unfortunately, PART does not generate expressions for parameters whose range is shared by the two classes under consideration (i.e., DNN failure and DNN correct). This is an issue as SEDE generates safe images that are similar to the unsafe ones. For example, the PART output in Fig. 4 does not include an expression with "$HeadPose_X > 10$", which indicates that the head is looking top because this characteristic is observed in both safe and unsafe images. Such commonalities, however, might be important to characterize unsafe inputs; indeed, in our example, if the RCC includes images with a person looking top, safe images close to the generated unsafe images will likely be looking top. To address this issue, SEDE joins the PART expression with an expression that constrains the parameters not appearing in the PART expression. The additional expression is generated by identifying the min and max values assigned to the parameters for all the unsafe images. For the example above, it joins the expression "$HeadPose_X > 10$" with the PART expression in Equation 12, thus leading to:

$$
\begin{aligned}
&HeadPose_X > 10 \& \\
&\quad ((HeadPose_Y > 50.34) \parallel \\
&\quad\quad (\neg(HeadPose_Y > 50.34) \& \neg(HeadPose_Y < 13.34) \\
&\quad\quad\quad \&(HeadPose_Z > 60 \& HeadPose_Y > 30)) \parallel \\
&\quad\quad (\neg(HeadPose_Y > 50.34) \& \neg(HeadPose_Y < 13.34) \\
&\quad\quad\quad \& \neg(HeadPose_Z > 60 \& HeadPose_Y > 30) \\
&\quad\quad\quad \& \neg(HeadPose_Z \leq 60)))
\end{aligned}
\tag{13}
$$

3.3 Step 4. Generate unsafe images for retraining

In Step 4, for each RCC, SEDE automatically generates a set U of unsafe images by relying on the simulator. Thanks to the simulator, these images can indeed be automatically labeled and, consequently, can be used to retrain the DNN. We refer to this set of images as the *unsafe improvement set*. More precisely, we aim to train a new DNN model by relying on an extended training set that consists of part of the training set used to train the DNN under analysis and the unsafe improvement set. The retraining process is configured as a fine-tuning process where the weights of the DNN to be trained are initially set to be equal to the weights of the DNN under analysis. Our

rationale is that by retraining the DNN with an additional set of images belonging to the unsafe portion of the input space, we enable the DNN to better learn how to provide correct outputs for such inputs.

To avoid losing information derived from the original training process, the extended training set consists of $S\%$ of the simulator images and $R\%$ of the real-world images belonging to the original training set. We suggest configuring S and R such that the number of simulator images is not overwhelmingly larger than the number of real-world images (e.g., twice the number of real-world images, at most). More precisely, we should limit the number of simulator images because they may overly influence the DNN with characteristics that are not present in real-world images (e.g., faces with regular shapes); nevertheless, we suggest keeping a representative set of simulator-based images (e.g., 5%) because they may include a number of scenarios (in our case studies, head positions) not covered by real-world images. For real-world images, since they are generally low in number, we suggest keeping most or all of them to prevent the risk of losing any image that captures an under-represented scenario. For our experiments, we set S=5% and R=100% (see Section 4.6). We leave to future work the identification of a solution for the automated selection of the best configuration for S and R (e.g., by selecting the DNN with the best accuracy among the ones generated by repeating the retraining process with different values for S and R). For example, recent results suggest that the proportion of simulator ($S\%$) and real-world ($R\%$) images may also depend on the degree of fidelity of the simulator: the more realistic simulator images are, the less real-world images are required [13].

The accuracy of the DNN training process may depend on a wide range of factors, including the specific simulator images selected from the original training set, the images belonging to the unsafe improvement set, and other random factors (e.g., the order of images in the training batches). For this reason, SEDE repeats the retraining process (i.e., generate unsafe images in U for each RCC and retrain the DNN) multiple times and keeps the DNN that provides the best output. Though repeating the training process obviously increases the training cost (e.g., buying additional GPU time in Cloud systems), for safety-critical systems, such additional cost may be justified by improvements in accuracy.

4   EMPIRICAL EVALUATION

Our empirical evaluation addresses the following research questions:

RQ1 *How does PaiR fare, compared to NSGA-II and DeepNSGA-II, for the generation of diverse images belonging to RCCs?* The generation of a sufficiently large and diverse set of images belonging to all the RCCs under analysis is essential to enable the characterization of DNN failures. To that end, SEDE integrates a dedicated genetic algorithm (PaiR) whose performance is evaluated by this RQ and compared to NSGA-II and DeepNSGA-II in terms of percentage of RCCs for which an individual has been generated, percentage of individuals belonging to each cluster, and image diversity within clusters.

RQ2 *Does SEDE generate simulator images that are close to the center of each RCC?* The generation of images belonging to a RCC is a necessary step to generate expressions that characterize the RCC. Since there is no guarantee of generating, using simulators, images that are similar to real-world images, this research question evaluates if the images generated by SEDE are closer to the center (medoid) of a cluster than random simulated, unsafe images.

RQ3 *Does SEDE generate, for each RCC, a set of images sharing similar characteristics?* To successfully characterize RCCs with PART, it is necessary, for each RCC, to rely on a set of images presenting similar characteristics while preserving diversity regarding other aspects. In

other words, the generated images shall present similar values for a subset of the simulator parameters while being diverse regarding the other parameters.

RQ4 *Do the RCC expressions identified by SEDE delimit an unsafe space?* This research question aims to determine if the analyzed DNN underperforms when processing images matching RCC expressions. In other words, is the DNN accuracy significantly lower for such images, as expected, than for random images from the whole input space?

RQ5 *How does SEDE compare to state-of-the-art DNN accuracy improvement practices?* This research question evaluates the effectiveness of SEDE (Step 4), in terms of accuracy improvements, compared to HUDD and a baseline consisting of randomly generated images for each RCC (namely, random baseline).

4.1 Subject Systems

We consider DNNs that implement head pose detection and face landmarks detection. They are building blocks for in-car monitoring systems (e.g., driver's drowsiness detection) under study at IEE Sensing, our industry partner.

The head pose detection DNN (HPD) receives as input the cropped image of the head of a person and determines its pose according to nine classes (straight, turned bottom-left, turned left, turned top-left, turned bottom-right, turned right, turned top-right, reclined, looking up).

The facial landmarks detection DNN (FLD) determines the location of the pixels corresponding to 27 face landmarks delimiting seven face elements: nose ridge, left eye, right eye, left brow, right brow, nose, and mouth. Several face landmarks match each face element (e.g., there are four landmarks to outline a mouth). The accuracy of this regression DNN is computed as the percentage of images with landmarks being accurately predicted. For each landmark, we compute the prediction error as the Euclidean distance between the predicted and correct landmark pixel on the input image; an image is considered accurately predicted if the average error is below 4 pixels, as suggested by IEE engineers.

Our DNNs have been trained with simulator images, fine-tuned with real-world images, and tested with real-world images, following the process in Fig. 6. For our experiments we relied on two different simulators developed by IEE and based on the 3D simulation of MakeHuman [10] and MBLab [44] using the rendering engine Blender [7]. The first simulator is called IEE-Faces, it relies on seven preset MakeHuman models of human faces developed by IEE; it provides 13 parameters that enable controlling different characteristics of the image such as illumination angle and head orientation. Table 1 provides a description of all the parameters of IEE-Faces, where the first three rows capture nine parameters, three for each row. IEE-Faces generates one image in 20 seconds, on the hardware used for our experiments. The second simulator is called IEE-Humans and it generates images that are more realistic (i.e., have more details) than the ones generated by IEE-Faces. IEE-Humans provides 23 different configuration parameters, which are described in Table 2 where the first two rows capture six parameters, three for each row. It takes 100 seconds to generate an image.

We trained three DNNs in total, two HPD DNNs and one FLD DNN. One HPD DNN (hereafter, HPD-F) has been trained using IEE-Faces and another one (hereafter, HPD-H) using IEE-Humans. The FLD DNN has been trained using IEE-Faces; we could not train FLD with IEE-Humans because it does not provide landmarks for the generated images.

To fine-tune and test the HPD DNNs we relied on the BIWI real-world dataset [18], which contains over 15, 000 pictures of 20 people faces (6 females and 14 males) annotated with head pose angle. People were recorded sitting in front of a Kinect [45], which is a motion sensor add-on for the Xbox 360 gaming console, and were asked to turn their head around trying to span all possible yaw/pitch angles they could perform. For each image, the head pose angle is computed using

Table 1. IEE-Faces simulator parameters

Parameter	Description
Camera Direction	Direction of scenario camera (X, Y, Z)
Camera Location	Location of scenario camera (X, Y, Z)
Lamp Location	Location of scenario lamp source (X, Y, Z)
$HeadPose_X$	Vertical position of the head (degrees)
$HeadPose_Y$	Horizontal position of the head (degrees)
$HeadPose_Z$	Tilting position of the head (degrees)
Makehuman Model	Preset makehuman face model (9 face models)

Table 2. IEE-Humans simulator parameters

Parameter	Description
Lamp Location	Location of scenario lamp source (X, Y, Z)
Lamp Direction	Direction of scenario lamp source (X, Y, Z)
Lamp Color	Color of scenario lamp source (R, G, B)
Camera Height	Height of scenario camera (pixels)
$HeadPose_X$	Vertical position of the head (degrees)
$HeadPose_Y$	Horizontal position of the head (degrees)
$HeadPose_Z$	Tilting position of the head (degrees)
Age	Age of the generated humanoid
Gender	Gender of the generated humanoid
Iris Size	Size of the humanoid iris
Pupil Size	Size of the humanoid pupil
Eye Saturation	Level of the humanoid eye saturation
Eye Color	Color of the humanoid eye
Eye Value	Controls the lightness level of iris
Skin Freckles	Amount of procedural freckles added to the skin
Skin Oil	Brightness of subtle oil effect on the skin
Skin Veins	Amount of procedural veins added to the skin

FaceShift [23]. For our experiments, we automatically labeled each image with a head pose class derived from the provided angles; we considered 1,000 close-up pictures— similar to what can be obtained with in-car, DNN-based sensors—belonging to two persons where one is used for training and the other one for testing. To generate close-up pictures we performed face detection and image cropping with the Dlib framework [61]. We could select only two persons from the BIWI dataset because the pictures belonging to other subjects were taken far from the camera and thus not useful to mimic a realistic situation with an in-car camera shooting a driver. Further, to simulate a real-world scenario where the DNN processes images of people never observed during training, we do not test the DNN using images belonging to the person used for training. To fine-tune and test the FLD DNN, since we require accurately annotated landmarks, we relied on a dataset provided by IEE.

Table 3 provides the number of images used to fine-tune the DNNs along with the obtained accuracy (i.e., the percentage of images for which the DNN provides correct outputs). Column *Data Source* provides the names of both the simulator and the real-world dataset used to train and fine-tune the DNNs. The columns under *Simulator-based Training* and *Fine-tuning* provide details about the size (i.e., number of images) of the training and test sets used in those phases along with the accuracy of the DNN. The data set used for training the fine-tuned DNNs (FLD, HPD-F, HPD-H) consists of simulator images (3, 000 for FLD, 21, 500 for HPD-F, 18, 000 for HPD-H) and real-world images (6, 000 for FLD, 476 for HPD-F, 476 for HPD-H); in Table 3, we report the size of the whole fine-tuning training set, including both simulator and real-world images.

Table 3. Case Study Systems

DNN	Data Source		Simulator-based Training			Fine-tuning			
	Simulator	Real-world	Training Set Size	Test Set Size	Epochs	Training Set Size	Simulator-based Test Set Size	Real-world Test Set Size	Epochs
			(Accuracy)	(Accuracy)		(Accuracy)	(Accuracy)	(Accuracy)	
FLD	IEE-Faces	IEE	16,000 (99.92%)	2,750 (44.41%)	10	9,000 (95.44%)	2,825 (43.22%)	1,000 (80.06%)	50
HPD-F	IEE-Faces	BIWI	21,500 (91.20%)	2,750 (85.43%)	18	21,976 (95.35%)	2,200 (87.10%)	500 (51.65%)	9
HPD-H	IEE-Humans	BIWI	15,400 (85.38%)	3,000 (80.21%)	25	18,476 (90.23%)	2,750 (85.23%)	500 (51.03%)	28

Table 4. SEDE Configuration

Step	Population Size	Number of iterations	Crossover probability	Mutation probability	Avg. Execution Time (hrs.)		
					FLD	HPD-F	HPD-H
2.1	25	100	0.7	0.3	40	40	200
2.2	25	100	0.3	0.3	20	20	100
2.3	25	100	0.3	0.3	20	20	100

Columns *Simulator-based Training – Epochs* and *Fine-tuning – Epochs* report the number of epochs considered to train and fine-tune the DNNs, respectively. All the DNNs have been fine-tuned for a number of epochs sufficient to achieve an accuracy above 90% with a training set of simulator and real-world images.

All the DNNs were implemented with PyTorch [55]. HPD follows the AlexNet architecture [39] which is commonly used for image classification tasks, while FLD follows the Hourglass architecture [51], which is optimized for landmarks detection (regression tasks).

In SEDE Step 1, for each case study DNN, we process all the failure-inducing images of the case study with HUDD to generate RCCs. HUDD's execution led to 10 RCCs for HPD-F DNN, 11 RCCs for HPD-H DNN, and 10 RCCs for FLD DNN.

Table 4 provides further information about the configuration of SEDE. Since in Steps 2.2 and 2.3 each individual of the initial population matches the reference one (i.e., the individual that should be similar to the one identified in the final population), we use low crossover and mutation probability to mutate only a few chromosomes at a time (i.e., we depart from the initial individual slowly).

4.2 RQ1. *How does PaiR fare, compared to NSGA-II and DeepNSGA-II, for the generation of diverse images belonging to RCCs?*

4.2.1 Experiment Design. We aim to demonstrate that PaiR, our dedicated algorithm to derive a diverse image population belonging to a RCC, performs better than NSGA-II and DeepNSGA-II. To this end, we measure the diversity in the populations generated by the three algorithms over time, for all the RCCs under analysis.

We choose NSGA-II as a baseline for comparison since its crowding distance function is designed to preserve and optimize the diversity between individuals by prioritizing individuals being more distant from others, along with individuals at the boundaries of the objective space.

We configured NSGA-II with an objective function (Equation 14) that drives the generation of individuals belonging to the RCC through their normalized distance from the cluster's medoid, as for PaiR (Equation 5).

$$F^C(i) = RCC_{distance}(C, i) \tag{14}$$

Additionally, we compare PaiR with DeepNSGA-II because the latter is a state-of-the-art solution to generate a population of individuals that are diverse. To this end, we extended the implementation provided for DeepJanus [58]. For our experiments, we rely on the following fitness functions:

$$F_1(i) = ChromosomeDistance(i, closest_{i_A}) \tag{15}$$

$$F_2^C(i) = RCC_{distance}(C, i) \tag{16}$$

F_1, consistent with DeepJanus and DeepMetis, is computed as the distance between individual i and the closest individual in the archive ($closest_{i_A}$). F_2^C measures the distance between an individual i and the medoid of the RCC C under analysis, thus enabling the algorithm to drive the search towards its objective: generating individuals that belong to C. We execute four independent runs of DeepNSGA-II for each RCC under analysis. At every search iteration, individuals are added to the archive if they belong to the RCC under analysis (i.e., $F_2^C(i) \le 1$); we consider a sparseness threshold of zero (i.e., we add to the archive any individual that differs from the ones already in the archive, which happens when $F_1(i) > 0$). Since, at the end of the search, the archive may contain a number of individuals larger than the one required for the next steps of the algorithm (i.e., 25, in our experiments), consistent with PaiR, we select the 25 individuals with the highest F_1.

For all three compared algorithms (NSGA-II, DeepNSGA-II, and PaiR), to measure the diversity of the population, since every individual is represented by a vector with the simulator parameter values used to generate the image (chromosome), we compute the average of the chromosome distance across pairs of individuals in the population. Individuals that do not belong to any cluster are ignored from the computation of diversity; indeed, such an individual might increase diversity but it would not be useful for SEDE.

We configured PaiR, NSGA-II, and DeepNSGA-II with the same time budget; precisely, we let NSGA-II and DeepNSGA-II execute for the duration required by our algorithm to perform 100 iterations. It amounts to 40 hours for HPD-F and FLD, 200 hours for HPD-H; differences are due to the time taken by the simulators to generate a single image. We executed PaiR, NSGA-II and DeepNSGA-II for all the 31 RCCs identified for our case study DNNs. To account for randomness, we ran the experiment four different times for each root-cause cluster. With a total of 31 RCCs, the four experiment runs led to the collection of 124 data points, thus enabling statistical analysis within practical execution time.

To compare the increase in diversity achieved over time by the three algorithms, for each RCC, we recorded the diversity achieved by the three algorithms every hour. Also, we tracked the percentage of individuals belonging to any RCC—a larger number of such individuals is expected to lead to better results in later stages of SEDE— and the percentage of covered clusters (i.e., RCCs with at least one individual belonging to them).

PaiR performs better than NSGA-II and DeepNSGA-II if, over time, the following conditions hold: (1) the diversity achieved by PaiR is significantly higher than that achieved by NSGA-II and DeepNSGA-II, which would indicate that PaiR fits better our purpose (i.e., generating a diverse population of individuals); (2) PaiR generates a larger proportion of individuals belonging to any RCC, which is useful to characterize RCCs since PART rules can be expected to be more accurate if a larger set of data points is considered; (3) PaiR covers a larger number of RCCs (i.e., PaiR generates

Table 5. RQ1: Average diversity across RCCs for HPD-F

Time (hrs.)	Avg. diversity (standard deviation)			Statistical test			
				NSGA-II		DeepNSGA-II	
	PaiR	NSGA-II	DeepNSGA-II	$p\text{-}value$ (U-test)	Effect Size ($\hat{A}_{12}$)	$p\text{-}value$ (U-test)	Effect Size ($\hat{A}_{12}$)
5	0.001 (0.002)	0.000 (0.000)	0.003 (0.010)	3.18e-03	0.60	0.159	0.555
10	0.010 (0.017)	0.014 (0.020)	0.007 (0.011)	0.870	0.51	0.190	0.576
15	0.010 (0.013)	0.002 (0.003)	0.011 (0.012)	8.53e-03	0.66	0.843	0.512
20	0.010 (0.014)	0.002 (0.002)	0.011 (0.013)	5.53e-04	0.71	0.843	0.512
25	0.012 (0.013)	0.002 (0.002)	0.015 (0.014)	6.18e-06	0.78	0.711	0.476
30	0.013 (0.014)	0.002 (0.002)	0.015 (0.014)	5.45e-05	0.75	0.771	0.481
35	0.013 (0.014)	0.002 (0.002)	0.015 (0.014)	6.18e-06	0.78	0.741	0.479
40	0.014 (0.015)	0.002 (0.003)	0.016 (0.014)	1.31e-05	0.77	0.640	0.470

at least one image belonging to each cluster), thus enabling the characterization of a larger number of RCCs in later SEDE steps.

4.2.2 Results. Fig. 9 shows the evolution of diversity obtained with PaiR, DeepNSGA-II, and NSGA-II for our case study DNNs. To simplify visual comparisons, we plot the average, minimum, and maximum diversity observed at each timestamp (every hour); data points are diversity values observed across the four executed runs for all RCCs.

Table 6. RQ1: Average diversity across RCCs for HPD-H

Time (hrs.)	Avg. diversity (standard deviation)			Statistical test			
				NSGA-II		DeepNSGA-II	
	PaiR	NSGA-II	DeepNSGA-II	$p\text{-}value$ (U-test)	Effect Size ($\hat{A}_{12}$)	$p\text{-}value$ (U-test)	Effect Size ($\hat{A}_{12}$)
5	0.028 (0.039)	0.028 (0.056)	0.011 (0.045)	0.062	0.612	2.58e-08	0.805
10	0.028 (0.039)	0.005 (0.008)	0.109 (0.083)	5.43e-03	0.669	2.45e-03	0.314
15	0.029 (0.037)	0.002 (0.002)	0.113 (0.077)	5.45e-10	0.880	4.12e-04	0.282
20	0.028 (0.035)	0.002 (0.002)	0.125 (0.067)	5.66e-09	0.860	1.80e-06	0.205
25	0.029 (0.036)	0.010 (0.022)	0.146 (0.061)	4.06e-06	0.785	4.32e-10	0.114
50	0.056 (0.049)	0.003 (0.002)	0.156 (0.047)	2.30e-12	0.934	3.09e-11	0.089
75	0.072 (0.061)	0.003 (0.003)	0.135 (0.014)	1.20e-13	0.959	5.37e-10	0.115
100	0.088 (0.063)	0.004 (0.002)	0.149 (0.015)	1.89e-15	0.992	1.54e-06	0.202
125	0.102 (0.071)	0.004 (0.003)	0.159 (0.012)	6.37e-16	1.0	8.51e-04	0.293
150	0.120 (0.083)	0.004 (0.004)	0.155 (0.020)	6.37e-16	1.0	0.021	0.357
175	0.140 (0.095)	0.003 (0.003)	0.157 (0.017)	6.37e-16	1.0	0.049	0.378
200	0.151 (0.093)	0.004 (0.004)	0.153 (0.016)	6.37e-16	1.0	0.369	0.444

In Fig. 9, we can observe that, after 13 hours of execution, both PaiR and DeepNSGA-II outperform NSGA-II; also, with the largest test budget (i.e., 200 hours for HPD-H and 40 hours for HPD-F and FLD) PaiR performs either similar to DeepNSGA-II (for HPD-F and HPD-H) or outperforms it (FLD). However, PaiR and DeepNSGA-II differ in performance across case studies: In the first 10 hours of execution, for the case study subjects relying on a low-fidelity simulator (HPD-F and FLD), PaiR shows better ($p\text{-}value \leq 0.05$ for FLD) or slightly better ($p\text{-}value > 0.05$ but $\hat{A}_{12} \geq 0.55$ for HPD-F) performance than DeepNSGA-II, while DeepNSGA-II yields better initial performance than PaiR for the case study with a high-fidelity simulator (HPD-H). We believe that such result can be explained by the fact that, since the low-fidelity simulator provides less configuration parameters, it is more difficult to generate diverse images than with a high-fidelity simulator. Therefore, with
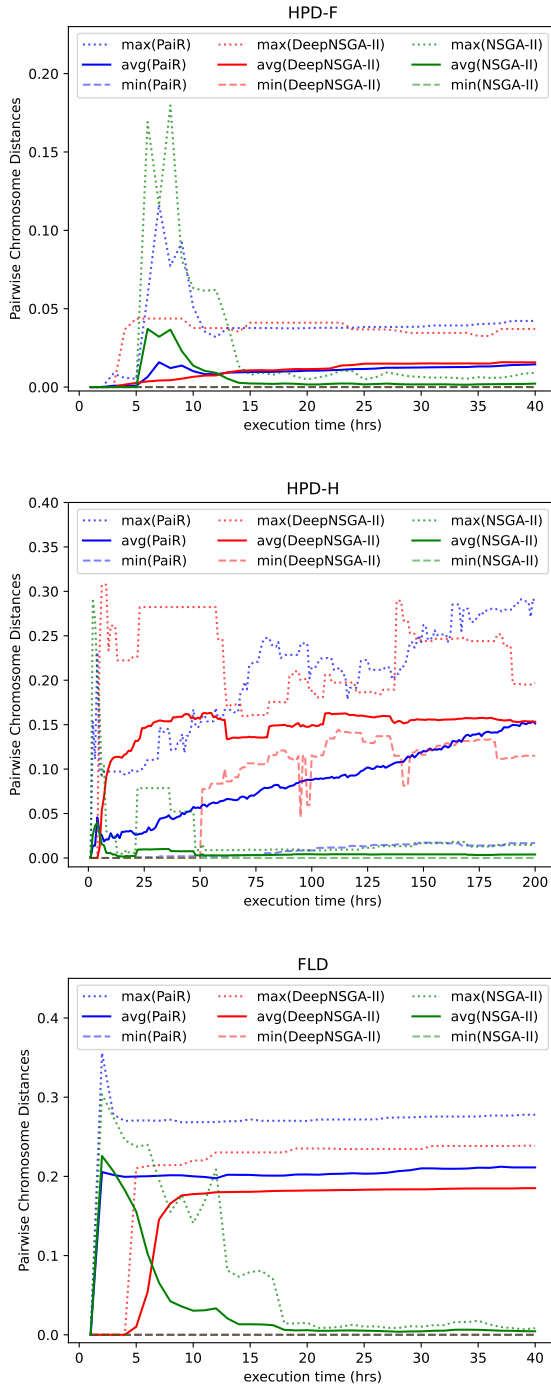
Fig. 9. Evolution of diversity achieved by PaiR compared to NSGA-II and DeepNSGA-II for HPD-F, HPD-H, and FLD DNNs

Table 7. RQ1: Average diversity across RCCs for FLD

| Time (hrs.) | Avg. diversity (standard deviation) | | | Statistical test | | | |
| | | | | NSGA-II | | DeepNSGA-II | |
	PaiR	NSGA-II	DeepNSGA-II	$p\text{-}value$ (U-test)	Effect Size ($\hat{A}_{12}$)	$p\text{-}value$ (U-test)	Effect Size ($\hat{A}_{12}$)
5	0.200 (0.077)	0.156 (0.073)	0.010 (0.045)	4.01e-03	0.685	6.06e-13	0.932
10	0.200 (0.077)	0.030 (0.044)	0.178 (0.061)	4.36e-10	0.905	0.015	0.657
15	0.202 (0.077)	0.013 (0.023)	0.181 (0.063)	4.36e-10	0.905	0.028	0.643
20	0.202 (0.077)	0.006 (0.004)	0.182 (0.063)	4.36e-10	0.905	0.013	0.660
25	0.203 (0.077)	0.005 (0.003)	0.183 (0.064)	4.36e-10	0.905	0.028	0.643
30	0.210 (0.080)	0.004 (0.003)	0.184 (0.064)	4.36e-10	0.905	2.08e-03	0.700
35	0.211 (0.080)	0.006 (0.005)	0.185 (0.065)	4.36e-10	0.905	1.60e-03	0.705
40	0.211 (0.080)	0.004 (0.002)	0.185 (0.065)	4.36e-10	0.905	1.60e-03	0.705

a low-fidelity simulator it might be easier to achieve higher diversity by continuously evolving the same population (as done by PaiR) rather than by relying on repopulation (what is integrated into DeepNSGA-II). Indeed, repopulation forces the algorithm to use the test budget to re-generate images belonging to the RCC rather than diversifying images already belonging to a RCC.

Further, we can observe that, during the first 10 hours of execution, both PaiR and NSGA-II present a sharp peak for both average and maximum diversity. This is mainly due to the fact that in the first hours of execution the two algorithms generate a limited number of images belonging to each RCC (see Fig. 10), thus leading to high diversity regardless of the generation strategy. After 10 hours of execution and the initial peak, for PaiR, we can observe an up-trend in average diversity reaching 0.014, 0.151, and 0.211 for HPD-F, HPD-H, and FLD, respectively. In contrast, NSGA-II presents an average diversity that keeps decreasing till 15 hours (HPD-F), 50 hours (HPD-H), and 20 hours (FLD), and then stabilizes around 0.002, 0.004, and 0.004, at much lower levels than PaiR. Such initial peak is not observed with DeepNSGA-II, likely because of repopulation. Indeed, repopulation slows down the identification of images belonging to a cluster: DeepNSGA-II requires between 10 and 75 hours to generate the same number of RCC images generated by PaiR in five hours, as shown in Tables 5 to 7, discussed below. However, repopulation leads to the identification of more diverse images since those generated by DeepNSGA-II in later iterations are not similar to the ones generated in previous iterations, as shown by the up-trending DeepNSGA-II curve. For DeepNSGA-II, we observe an average diversity reaching 0.016 (HPD-F), 0.153 (HPD-H), and 0.185 (FLD), respectively.

Tables 5, 6, and 7 report statistics about the diversity obtained for all RCCs, for PaiR, NSGA-II, and DeepNSGA-II along with the standard deviation. To compare the three techniques, we also provide the Vargha and Delaney's $\hat{A}_{12}$ effect size and p-values resulting from a non-parametric Mann-Whitney U-test, where each individual observation is the diversity for one RCC in one of the four runs. PaiR achieves a significantly higher diversity ($p\text{-}value \leq 0.05$) compared to NSGA-II in all subjects and for all but two timestamps. For HPD-F, PaiR performs similarly to NSGA-II up to 10 hours of execution but then outperforms it with a large effect size (i.e., > 0.714)[5]. Given the safety-critical contexts in which the DNN under analysis should be adopted, we believe a budget of 20 hours to be acceptable for DNN analysis. PaiR performs similarly to DeepNSGA-II (i.e., $p\text{-}value > 0.05$), with a higher average diversity for PaiR up to 10 hours of execution.

For HPD-H, PaiR significantly outperforms NSGA-II at all timestamps except one (i.e., $p\text{-}value >$ 0.05 for 5-hour execution). However, DeepNSGA-II performs significantly better than PaiR for all

---

[5]Please note that effect size is considered small when $\hat{A}_{12} > 0.556$, medium when $0.638 < \hat{A}_{12} < 0.714$, large when $\hat{A}_{12} \geq 0.714$ [38].

Table 8. RQ1: Percentage of individuals belonging to RCCs for HPD-F

Time (hrs.)	Avg. (standard deviation) % of individuals belonging to RCCs			Statistical test			
				NSGA-II		DeepNSGA-II	
	PaiR	NSGA-II	DeepNSGA-II	*p-value* (U-test)	Effect Size ($\hat{A}_{12}$)	*p-value* (U-test)	Effect Size ($\hat{A}_{12}$)
5	6.7% (14.162)	0.0% (0.0)	0.60% (2.134)	2.06e-04	0.65	8.72e-03	0.616
10	42.0% (47.27)	32.0% (44.36)	10.2% (22.72)	0.309	0.56	7.01e-03	0.656
15	60.0% (49.61)	40.0% (49.61)	25.7% (33.77)	0.076	0.60	7.48e-04	0.705
20	60.0% (49.61)	40.0% (49.61)	31.7% (37.10)	0.076	0.60	7.48e-04	0.705
25	60.7% (48.82)	40.0% (49.61)	38.9% (38.84)	0.024	0.63	9.77e-04	0.706
30	70.0% (46.41)	40.0% (49.61)	42.1% (40.36)	7.48e-03	0.65	2.10e-05	0.764
35	70.0% (46.41)	40.0% (49.61)	44.5% (40.98)	7.48e-03	0.65	2.09e-05	0.764
40	70.0% (46.41)	40.0% (49.61)	45.2% (41.17)	7.48e-03	0.65	2.88e-05	0.760

Table 9. RQ1: Percentage of individuals belonging to RCCs for HPD-H

Time (hrs.)	Avg. (standard deviation) % of individuals belonging to RCCs			Statistical test			
				NSGA-II		DeepNSGA-II	
	PaiR	NSGA-II	DeepNSGA-II	*p-value* (U-test)	Effect Size ($\hat{A}_{12}$)	*p-value* (U-test)	Effect Size ($\hat{A}_{12}$)
5	50.91% (39.87)	27.27% (35.60)	0.91% (2.84)	1.24e-03	0.694	2.26e-12	0.909
10	70.91% (40.80)	41.09% (42.93)	8.55% (4.69)	3.75e-06	0.776	8.78e-08	0.824
15	85.45% (32.38)	45.45% (43.00)	9.45% (4.32)	1.24e-10	0.880	6.01e-12	0.909
20	90.91% (23.41)	49.09% (40.58)	10.27% (4.26)	6.38e-12	0.909	4.36e-17	1.0
25	92.73% (23.26)	59.27% (36.63)	10.91% (3.89)	2.71e-13	0.929	1.48e-17	1.0
50	98.18% (5.82)	66.18% (35.71)	20.64% (19.92)	1.87e-14	0.950	3.41e-17	1.0
75	100.0% (0.0)	66.18% (35.71)	61.09% (34.37)	5.02e-18	1.0	5.39e-18	1.0
100	100.0% (0.0)	68.00% (32.60)	71.36% (26.34)	5.14e-18	1.0	5.48e-18	1.0
125	100.0% (0.0)	74.55% (29.29)	73.36% (24.22)	4.44e-18	1.0	5.38e-18	1.0
150	100.0% (0.0)	74.55% (29.29)	77.91% (18.79)	4.44e-18	1.0	5.24e-18	1.0
175	100.0% (0.0)	74.55% (29.29)	82.55% (13.89)	4.44e-18	1.0	4.92e-18	1.0
200	100.0% (0.0)	74.55% (29.29)	84.36% (11.60)	4.44e-18	1.0	4.76e-18	1.0

the timestamps except two. Indeed, at 5 hours of execution PaiR performs significantly better than DeepNSGA-II with a large effect size and converges to a similar diversity after 200 hours.

For FLD, PaiR performs significantly better than both NSGA-II and DeepNSGA-II, with large and medium effect sizes, respectively. Compared to HPD-F, which relies on the same simulator used by FLD, the performance of PaiR could be attributed to the ease of generating images belonging to FLD clusters; indeed, the radius of FLD clusters range between [27.33, 74.65] while this range for HPD-F is [0.039, 0.145]. In such situation, PaiR can spend most of the test budget to maximize the diversity of the generated images.

Fig. 10 shows the average, minimum, and maximum percentage of individuals belonging to one of the RCCs under analysis, observed at each timestamp over the four executed runs. For PaiR, NSGA-II and DeepNSGA-II, the number of individuals belonging to any RCC increases over time, though this trend is much steeper for PaiR. Indeed, on average, a larger proportion of the individuals generated by PaiR belongs to a RCC for a given execution time. This should result in better supporting the generation of PART rules at later stages of SEDE.

Tables 8, 9, and 10 report statistics about the percentage of individuals belonging to any RCC, for PaiR, NSGA-II and DeepNSGA-II along with the standard deviation. We report the $\hat{A}_{12}$ statistics
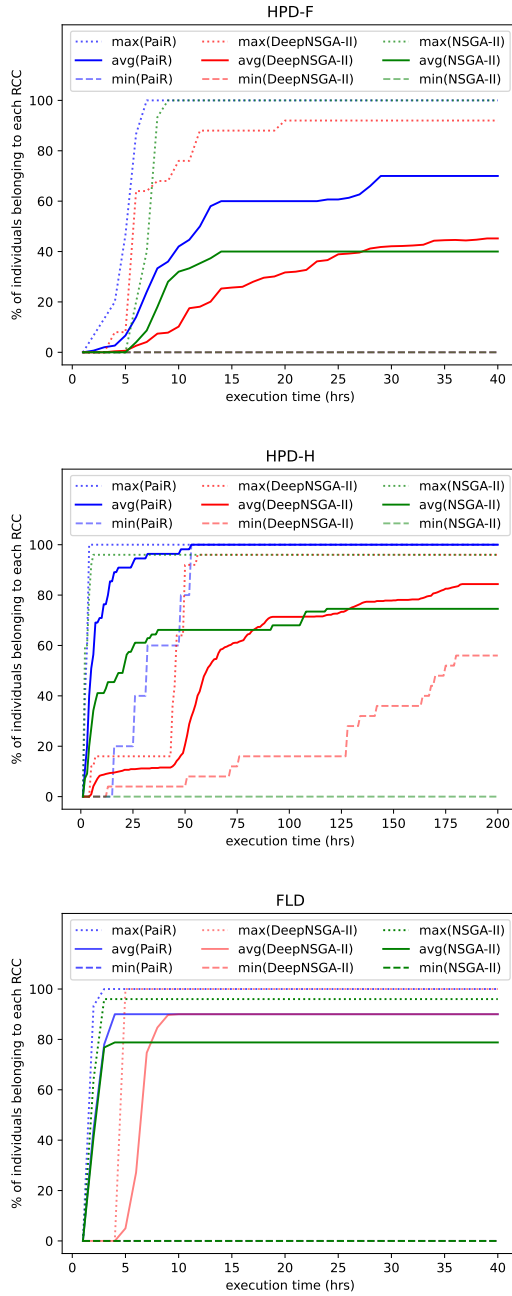
Fig. 10. Percentage of individuals belonging to a cluster generated by PaiR compared to NSGA-II and DeepNSGA-II for HPD-F, HPD-H, and FLD DNNs

Table 10. RQ1: Percentage of individuals belonging to RCCs for FLD

| Time (hrs.) | Avg. (standard deviation) % of individuals belonging to RCCs | | | Statistical test | | | |
| | | | | NSGA-II | | DeepNSGA-II | |
	PaiR	NSGA-II	DeepNSGA-II	p-value (U-test)	Effect Size ($\hat{A}_{12}$)	p-value (U-test)	Effect Size ($\hat{A}_{12}$)
5	90.0% (30.38)	78.80% (28.13)	5.0% (22.07)	5.42e-11	0.905	4.06e-14	0.925
10	90.0% (30.38)	78.80% (28.13)	90.0% (30.38)	5.42e-11	0.905	1.0	0.5
15	90.0% (30.38)	78.80% (28.13)	90.0% (30.38)	5.42e-11	0.905	1.0	0.5
20	90.0% (30.38)	78.80% (28.13)	90.0% (30.38)	5.42e-11	0.905	1.0	0.5
25	90.0% (30.38)	78.80% (28.13)	90.0% (30.38)	5.42e-11	0.905	1.0	0.5
30	90.0% (30.38)	78.80% (28.13)	90.0% (30.38)	5.42e-11	0.905	1.0	0.5
35	90.0% (30.38)	78.80% (28.13)	90.0% (30.38)	5.42e-11	0.905	1.0	0.5
40	90.0% (30.38)	78.80% (28.13)	90.0% (30.38)	5.42e-11	0.905	1.0	0.5

and p-values resulting from a non-parametric Mann-Whitney U-test, where each observation is the percentage of individuals belonging to one cluster in one of the four runs. PaiR achieves a significantly higher number of individuals belonging to RCCs (p-value ≤ 0.05) compared to NSGA-II, for all subjects and timestamps except three. Compared to DeepNSGA-II, PaiR identifies a significantly higher number of individuals belonging to RCCs for all the cases except FLD, where both techniques fare similarly. For HPD-F, around 10 hours of execution, PaiR performs similarly to NSGA-II; both PaiR and NSGA-II perform significantly better than DeepNSGA-II. However, with a time budget of 25 hours PaiR performs significantly better than both NSGA-II and DeepNSGA-II, with an effect size above 0.60. For HPD-H, PaiR significantly outperforms NSGA-II and DeepNSGA-II (p-value ≤ 0.05); effect size is always large (above 0.78 in our cases) except for one case (medium, for 5-hour execution with NSGA-II). For HPD-F and HPD-H, we conjecture that DeepNSGA-II performs worse than PaiR because the images in a RCC are very similar to each other and several mutations of the best images in a population are needed to generate additional images belonging to a same RCC; such observation is supported by the plot for HPD-F, where PaiR lays in an intermediate plateau before reaching its best average result. The repopulation operator adopted by DeepNSGA-II may prevent DeepNSGA-II from generating individuals that belong to the RCC and are different from the already generated ones (e.g., after repopulation, DeepNSGA-II end-ups with images that match the ones already generated because they are simpler to generate). For FLD, PaiR quickly reaches (in less than 5 hours) a plateau for the average (90%) and max (100%) percentages of individuals per RCC. Though NSGA-II reaches its plateau in less than 5 hours, its plateau is lower than PaiR's (78.8% vs. 90.0%). Further, DeepNSGA-II reaches the same plateau values as PaiR but requires more time (i.e., 8 hours for the peak of the average curve). These results are mainly due to the ease of generating images belonging to most FLD clusters; indeed, all three algorithms quickly reach their peaks. However, both PaiR and DeepNSGA-II do not improve over 90% because they do not cover one RCC. In other words, for FLD, when PaiR and DeepNSGA-II analyse a cluster that they can cover, both generate a population of images that fully belongs to the cluster. The same observation cannot be made for NSGA-II; indeed, although all three algorithms cover the same FLD clusters, PaiR and DeepNSGA-II achieve a higher percentage of individuals per cluster. NSGA-II probably gets stuck in local optima.

Fig. 11 shows the percentage of clusters covered by PaiR, NSGA-II and DeepNSGA-II, for all subjects. PaiR is capable of covering (generating representative images of) a larger number of RCCs (27, in total): 7 out of 10 RCCs (70.0%) for HPD-F, 11 out of 11 RCCs (100.0%) for HPD-H, and 9 out of 10 RCCs (90.0%) for FLD. DeepNSGA-II covers a lower number of clusters (26, in total): 6 out of

10 RCCs (60.0%) for HPD-F, 11 out of 11 RCCs (100.0%) for HPD-H, and 9 out of 10 RCCs (90.0%) for FLD. NSGA-II, instead, covers only 23 RCCs: 4 out of 10 RCCs (40.0%) for HPD-F, 10 out of 11 RCCs (90.9%) for HPD-H, and 9 out of 10 RCCs (90.0%) for FLD. Since the cases in which PaiR does not cover all the clusters are the ones involving a simulator with a lower number of configuration parameters (i.e., HPD-F and FLD), we believe that our imperfect results are due to our limited control of the simulator in use. However, PaiR still performs better than NSGA-II and DeepNSGA-II, thus showing that it can better leverage the capabilities of the simulator. We conclude that PaiR is a better choice than both DeepNSGA-II and NSGA-II since it helps explain a larger number of RCCs, 27 (87.1%) compared to 26 (83.8%) with DeepNSGA-II and 23 (74.1%) with NSGA-II.
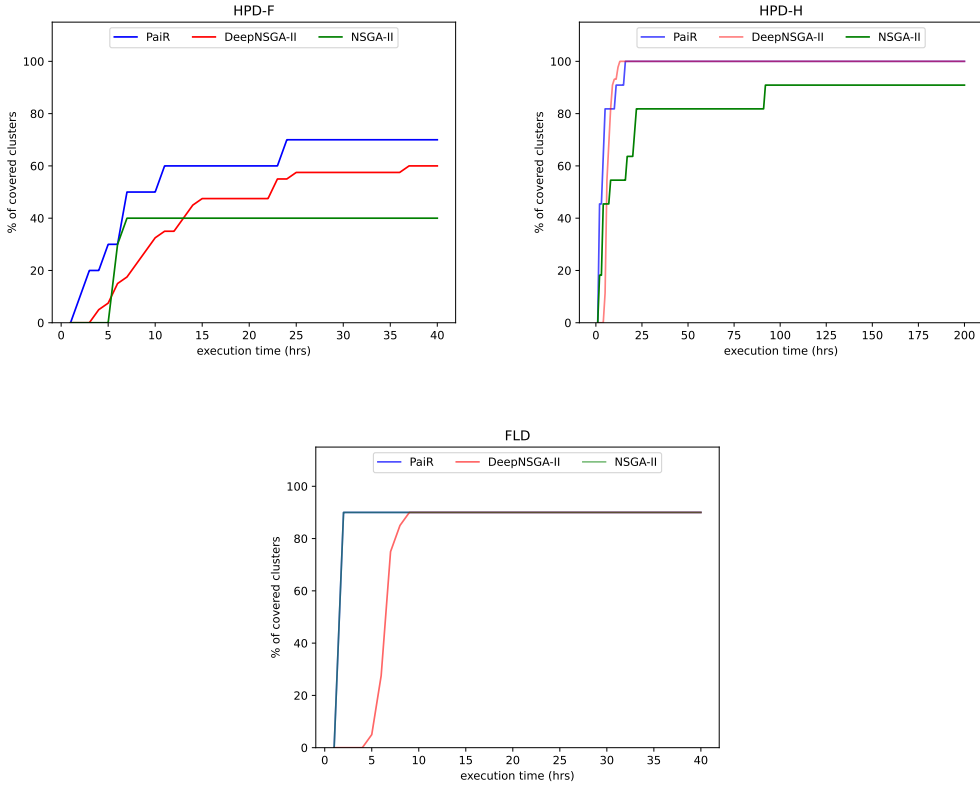

Fig. 11. Percentage of clusters covered by PaiR, NSGA-II and DeepNSGA-II for HPD-F, HPD-H, and FLD DNNs

Tables 11, 12, and 13 report the percentage of covered RCCs across the four runs, for PaiR, NSGA-II and DeepNSGA-II. Further, we report the p-value computed with a non-parametric Fisher's exact test where we compare the number of clusters covered by PaiR and the competing approaches. Unsurprisingly, there is no significant difference for FLD, where the three algorithms cover almost all the clusters because of the easiness of the task (see discussion above). Differences with DeepNSGA-II tend to be not significant, which indicates that covering a cluster is a simple task for both algorithms. However, differences are always significant for small time budget; indeed, up to 5 hours, PaiR covers

Table 11. RQ1: Percentage of covered RCCs for HPD-F

Time (hrs.)	Avg. % of covered RCCs			p-value (Fisher's Exact)	
	PaiR	NSGA-II	DeepNSGA-II	NSGA-II	DeepNSGA-II
5	30.0%	0.0%	7.5%	1.85e-04	0.019
10	50.0%	40.0%	32.5%	0.5	0.172
15	60.0%	40.0%	47.5%	0.117	0.369
20	60.0%	40.0%	47.5%	0.117	0.369
25	70.0%	40.0%	57.5%	0.012	0.352
30	70.0%	40.0%	57.5%	0.012	0.352
35	70.0%	40.0%	57.5%	0.012	0.352
40	70.0%	40.0%	60.0%	0.012	0.482

Table 12. RQ1: Percentage of covered RCCs for HPD-H

Time (hrs.)	Avg. % of covered RCCs			p-value (Fisher's Exact)	
	PaiR	NSGA-II	DeepNSGA-II	NSGA-II	DeepNSGA-II
5	81.82%	45.45%	11.36%	7.54e-04	1.84e-11
10	81.82%	54.55%	93.18%	0.011	0.195
15	90.91%	54.55%	100.0%	2.27e-04	0.116
20	100.0%	63.64%	100.0%	5.75e-06	1.0
25	100.0%	81.82%	100.0%	5.51e-03	1.0
50	100.0%	81.82%	100.0%	5.51e-03	1.0
75	100.0%	81.82%	100.0%	5.51e-03	1.0
100	100.0%	90.91%	100.0%	0.116	1.0
125	100.0%	90.91%	100.0%	0.116	1.0
150	100.0%	90.91%	100.0%	0.116	1.0
175	100.0%	90.91%	100.0%	0.116	1.0
200	100.0%	90.91%	100.0%	0.116	1.0

Table 13. RQ1: Percentage of covered RCCs for FLD

Time (hrs.)	Avg. % of covered RCCs			p-value (Fisher's Exact)	
	PaiR	NSGA-II	DeepNSGA-II	NSGA-II	DeepNSGA-II
5	90.0%	90.0%	5.0%	1.0	1.47e-15
10	90.0%	90.0%	90.0%	1.0	1.0
15	90.0%	90.0%	90.0%	1.0	1.0
20	90.0%	90.0%	90.0%	1.0	1.0
25	90.0%	90.0%	90.0%	1.0	1.0
30	90.0%	90.0%	90.0%	1.0	1.0
35	90.0%	90.0%	90.0%	1.0	1.0
40	90.0%	90.0%	90.0%	1.0	1.0

more clusters. A higher number of significant differences is instead observed when comparing PaiR to NSGA-II.

To summarize, our results suggest that the adoption of PaiR is the best choice in our context; indeed, PaiR can (1) generate images for a larger number of RCCs than NSGA-II and DeepNSGA-II, (2) generate significantly more images belonging to each RCC, compared to both DeepNSGA-II and NSGA-II, and, (3) achieve significantly higher image diversity than NSGA-II and similar diversity to that of DeepNSGA-II, except for one case study, where PaiR outperforms DeepNSGA-II.

In the rest of our evaluation, we ignore the clusters for which PaiR was not able to identify representative images (Step 2.1), as described above. Also, we ignore one cluster for which SEDE did

Table 14. RQ2: Average distance from medoid for HPD-F

RCC	Avg. distance from medoid		$p\text{-}value$ (U-test)	Effect Size ($\hat{A}_{12}$)
	SEDE Images	Unsafe Images		
1	0.075	0.469	5.11e-17	1.0
2	0.118	0.480	4.52e-16	1.0
3	0.058	0.476	7.17e-17	1.0
4	0.072	0.459	2.12e-16	1.0
5	0.085	0.457	6.45e-16	1.0
6	0.128	0.470	1.48e-16	1.0
7	0.096	0.439	5.43e-17	1.0

not identify unsafe images (Step 2.2), possibly because the hazard-triggering event is the presence of jewelry, which is not generated by our simulator. In total, we ignore three clusters in the case of HPD-F, and one cluster for HPD-H and FLD.

4.3 RQ2. *Does SEDE generate simulator images that are close to the center of each RCC?*

4.3.1 Experiment Design. This research question evaluates whether the images generated by SEDE for each RCC are closer to the medoid of the RCC than random failing images. Otherwise, we cannot claim that SEDE contributes to generating images that help characterize the RCC.

To measure the distance of an image from the RCC medoid we once again rely on the heatmap distance between the RCC medoid and the image (Equation 2).

For each RCC, we compute the distance between the RCC medoid and every image generated by SEDE to characterize the RCC in Step 2.1. Also, we compute the distance of the unsafe images in the simulator-based test set (i.e., random failing images) from the RCC medoid. To positively answer our research question, for each RCC, the average distance from the medoid to the images generated by SEDE should be significantly smaller than the average distance obtained with random unsafe simulated images, based on a non-parametric Mann-Whitney U-test.

4.3.2 Results. Fig. 12 shows boxplots reporting, for all RCCs across DNNs, the heatmap distances from RCC medoids, for images in the unsafe test set and those generated by SEDE for every RCC in Step 2.1. Fig. 12 shows that the images generated by SEDE (i.e., boxplots with IDs 1 to 26) are much closer to the RCC medoid than random unsafe images (i.e., boxplots with IDs UI-1 to UI-26). Indeed, with SEDE, the median lays in the ranges [0.056, 0.125] for HPD-F, [0.048, 0.113] for HPD-H, and [18.6, 64.3] for FLD. For random unsafe images, the median tends to be much higher and lays in the ranges [0.38, 0.48] for HPD-F, [0.08, 0.24] for HPD-H, and [42.0, 73.6] for FLD.

Tables 14 to 16 provide, for each RCC across DNNs, the average distance of SEDE images and random unsafe images, along with p-values and effect size.

On average, SEDE images are closer to the medoid of the clusters by 80.7% for HPD-F, 54.6% for HPD-H and 24.9% for FLD. These differences with unsafe test set images are always statistically significant (*p-value* $\leq$ 0.05) with a large effect size.

4.4 RQ3. *Does SEDE generate, for each RCC, a set of images sharing similar characteristics?*

4.4.1 Experiment Design. This research question assesses if the images generated by SEDE for each RCC, present similar characteristics (i.e., similar values for a subset of the simulator parameters). If this is true, for each RCC, we should observe a subset of parameters with a variance that is
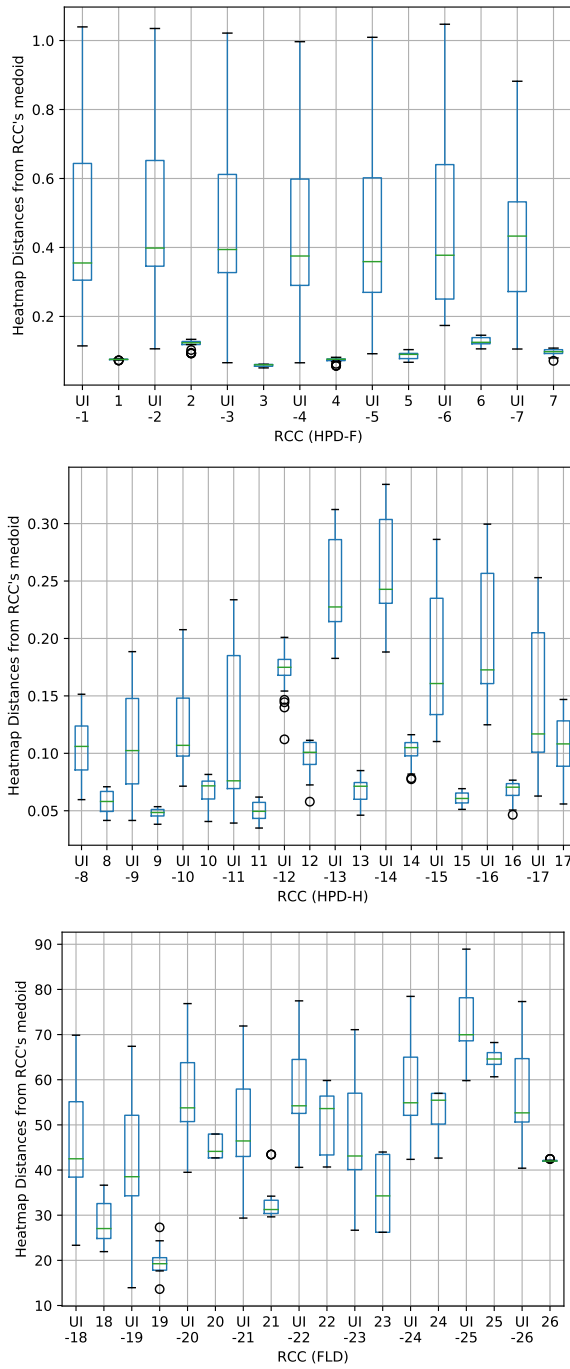
Fig. 12. Heatmap distances of SEDE images from the medoid of HPD-F (top), HPD-H (middle) and FLD (bottom) RCCs compared to unsafe test set images (UI)

Table 15. RQ2: Average distance from medoid for HPD-H

RCC	Avg. distance from medoid		*p-value* (U-test)	Effect Size ($\hat{A}_{12}$)
	SEDE Images	Unsafe Images		
8	0.057	0.135	1.52e-14	1.0
9	0.047	0.120	3.47e-14	1.0
10	0.067	0.134	1.43e-08	0.98
11	0.050	0.110	5.61e-12	1.0
12	0.098	0.183	4.81e-17	1.0
13	0.068	0.208	3.33e-17	1.0
14	0.101	0.214	1.13e-13	1.0
15	0.060	0.156	2.28e-17	1.0
16	0.067	0.159	7.11e-17	1.0
17	0.107	0.126	5.43e-03	0.91

Table 16. RQ2: Average distance from medoid for FLD

RCC	Avg. distance from medoid		*p-value* (U-test)	Effect Size ($\hat{A}_{12}$)
	SEDE Images	Unsafe Images		
18	25.66	46.28	1.79e-08	0.985
19	20.03	41.90	2.87e-10	1.0
20	45.17	56.84	4.12e-03	0.910
21	32.95	49.99	1.22e-05	0.965
22	49.99	57.64	0.037	0.845
23	35.63	48.03	3.84e-03	0.905
24	51.10	58.63	0.048	0.645
25	64.45	72.75	5.12e-04	0.940
26	42.11	53.30	4.32e-04	0.945

significantly lower than the variance observed in randomly generated images. For each RCC, instead of comparing the variance of each parameter p, which depends on the parameter range, we can compare the variance reduction rate (*VRR*), which can be computed as the ratio of the variance for a parameter p for a given RCC C_i over that of a set of randomly generated images:

$$VRR_{C_i}(p) = 1 - \frac{variance\ of\ p\ for\ the\ images\ in\ C_i}{variance\ of\ p\ for\ a\ set\ of\ random\ images}$$

For a parameter, a positive *VRR* indicates that its values are likely to be constrained to a smaller range than the one of the input domain. A 0.5 *VRR* means that the variance is reduced by 50%, which is considered to be a high reduction rate [17].

For each RCC, we compute *VRR* for each parameter. We positively answer our research question if, for a large number of RCCs, a subset of parameters presents a large *VRR* (> 0.5).

4.4.2 Results. Fig. 13 provides boxplots capturing the distribution of the variance reduction for each of the 26 RCCs. Each data point in a boxplot captures the variance reduction of one parameter.

Fig. 13 shows that for all the RCCs except boxplot 17, the top whisker is above 0.8; since the top whisker reports the max value (excluding outliers[6]) observed for a parameter, such result indicates that for all RCCs except one, we can observe at least one parameter with a high variance reduction,

---

[6]In our boxplots, $upperwhisker = min(max(x), Q_3 + 1.5 * IQR)$, $lowerwhisker = max(min(x), Q_1 - 1.5 * IQR))$, with IQR being the Inter Quartile Range and Q_x the x-th quantile.
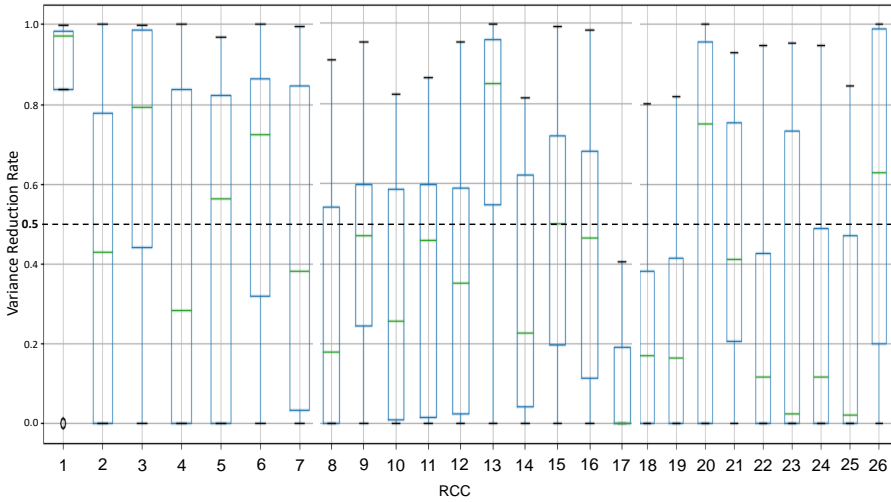
Fig. 13. Variance reduction for all simulator parameters associated with generated images in the root-cause clusters of HPD-F (boxplots 1 to 7), HPD-H (boxplots 8 to 17), and FLD (boxplots 18 to 26) DNNs

thus enabling us to positively answer our research question. Also, for 20 out of 26 clusters, the third quartile is above 0.5. Since it indicates the lowest value for the 25% data points having the highest variance reduction, it means that in 20 RCCs, 25% of the parameters present a variance reduction rate above 0.5. Therefore, for a large proportion of RCCs (77%), more than one parameter present a large variance reduction, which indicates that the hazard-triggering event is captured by several parameters (i.e., a specific combination of parameter values is needed to make the DNN fail).

4.5   RQ4. *Do the RCC expressions identified by SEDE delimit an unsafe space?*

4.5.1   Experiment Design. For each RCC, SEDE provides a set of expressions representing conjunctions of parameter ranges that characterize the unsafe images in the RCC. This research question evaluates if these expressions actually characterize an unsafe space, which implies that images whose parameters match such an expression are more likely to trigger a DNN failure than those that do not.

To address this research question, for each RCC, we generated 500 images matching the RCC expression and computed the DNN accuracy obtained with these images. To generate each image, for each simulator parameter, we selected a random value in the range provided in the expression. Also, we considered a random input set consisting of 500 images selected from the randomly generated simulator images used to test the fine-tuned DNN (see column Simulator-based Test Set in Table 3).

We can positively answer our research question if, for a large subset of RCCs, the images generated from RCC expressions lead to a significantly lower DNN accuracy[7] than randomly generated images, for each RCC. Since, for each RCC, we compare two image groups (i.e., 500 random images and 500 images generated by SEDE) labeled with a categorical variable (i.e., indicating if the DNN produces a correct output), we rely on the Fisher's exact test to assess differences in proportions of images leading to DNN failures.

---

[7]Recall that accuracy is 100% minus the percentage of inputs leading to DNN failures (i.e., if 90% of the inputs generated by SEDE is failure-inducing, we will observe an accuracy of 10%).

4.5.2   Results. Fig. 14 provides boxplots capturing the accuracy obtained with the random test set images and those generated according to RCC expressions. Each data point in the *SEDE images* boxplots corresponds to the DNN accuracy of one of the 26 RCC expressions. The accuracy observed with the random test sets for HPD-F, HPD-H, and FLD DNNs is 87.0%, 85.2%, and 43.2%, respectively. In contrast, the images generated according to RCC expressions lead to much lower accuracy in the ranges [6.2%, 79.8%], [34.0%, 66.6%], and [2.0%, 39.2%], for HPD-F, HPD-H and FLD, respectively. Moreover, for 25 out of 26 clusters, SEDE generates images leading to an accuracy that, based on a Fisher's exact test (see Tables 17, 18, and 19), significantly differs from the accuracy obtained with random images. Since in 25 out of 26 RCCs (96.15%) the RCC expressions generated by SEDE clearly delimit an unsafe space, we can positively answer RQ4.
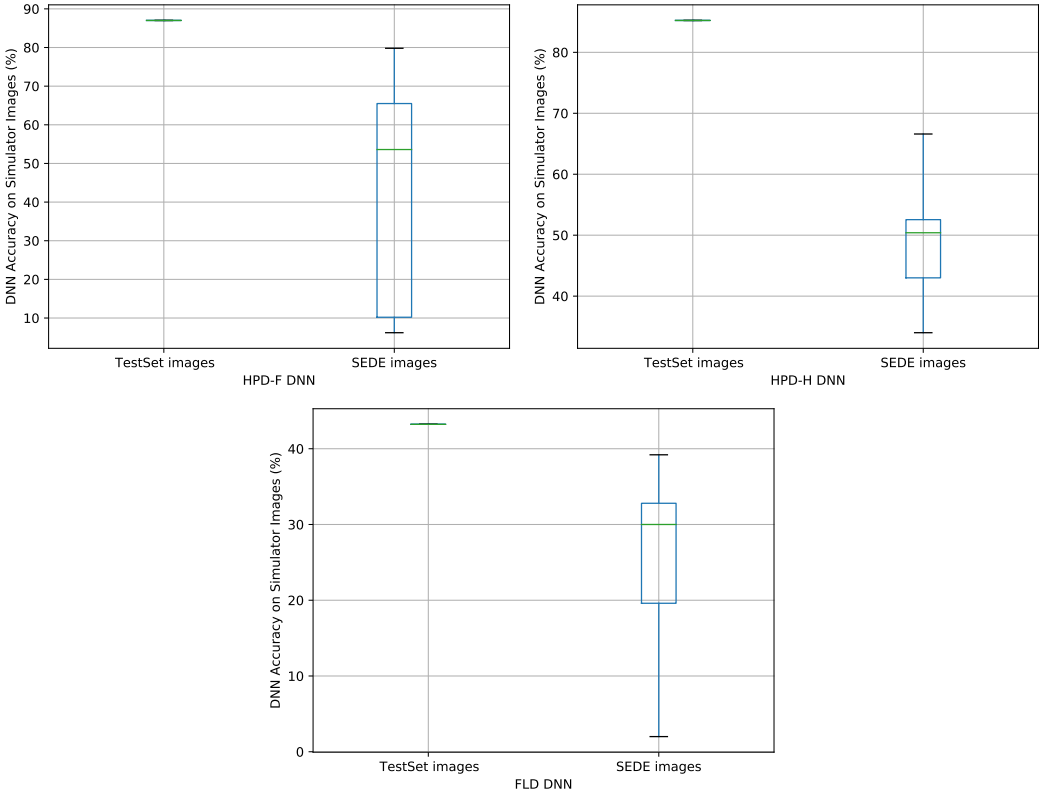

Fig. 14. Percentage of correctly classified images observed on SEDE images compared to the random test set images for HPD-F, HPD-H, and FLD DNNs

4.6   RQ5. *How does SEDE compare to state-of-the-art DNN accuracy improvement practices?*

4.6.1   Experiment Design. This research question aims to determine if SEDE performs better than state-of-the-art solutions. We compare SEDE with HUDD since it is the only approach that works with real-world images and addresses both root cause explanations and retraining (Section 5). Further, other retraining approaches [15, 16, 21, 35, 65, 66] generate retraining images through pixel value transformation (e.g., image contrast, image brightness, image blur, and image noise),

Table 17. RQ4: HPD-F accuracy within the unsafe space identified by SEDE compared to the whole input space

RCC	DNN Accuracy		p-value (Fisher's Exact)
	SEDE Unsafe Set	Random Input Set	
1	10.2%		7.40e-73
2	67.4%		9.25e-04
3	6.2%		9.11e-88
4	79.8%	87.0%	0.047
5	73.0%		0.019
6	59.8%		2.28e-06
7	53.6%		2.43e-09

Table 18. RQ4: HPD-H accuracy within the unsafe space identified by SEDE compared to the whole input space

RCC	DNN Accuracy		p-value (Fisher's Exact)
	SEDE Unsafe Set	Random Input Set	
8	39.4%		1.26e-19
9	50.6%		1.19e-10
10	34.0%		4.84e-25
11	49.0%		1.31e-11
12	41.0%		7.47e-18
13	51.8%	85.2%	5.66e-10
14	66.6%		1.73e-03
15	52.8%		2.19e-09
16	56.6%		2.32e-07
17	50.2%		7.14e-10

Table 19. RQ4: FLD accuracy within the unsafe space identified by SEDE compared to the whole input space

RCC	DNN Accuracy		p-value (Fisher's Exact)
	SEDE Unsafe Set	Random Input Set	
18	39.2%		0.30
19	19.6%		4.77e-13
20	30.0%		9.19e-88
21	32.8%		4.88e-03
22	19.6%	43.2%	4.39e-13
23	32.8%		4.12e-03
24	27.0%		2.82e-06
25	2.0%		2.49e-58
26	31.0%		6.31e-04

affine transformation (e.g., image translation, image scaling, image shearing, and image rotation), or adversarial modification (e.g., FGSM [24], PGD [43], C&W [8]). None of these approaches, however, aims to generate images similar to real-world, failure-inducing ones targeted by SEDE. In addition, DNNs might be inaccurate because either trained for a limited number of epochs or with a limited number of inputs; in such cases, retraining the DNN for additional epochs with a larger set of randomly selected inputs would suffice. For this reason, we consider a baseline approach which consists of retraining the DNN with an additional set of randomly generated images.

Regarding SEDE, we followed the procedure described in Section 3.3. The third column of Table 20 provides the size of the training set considered to retrain each DNN; precisely, we report the total number of simulator images generated according to SEDE expressions (*Unsafe Set*), the number of simulator images retained from the simulator training set (*Training Set Sim.*) and the number of real-world images retained from the set used for fine-tuning (*Training Set Real*). For each RCC, we have generated, with SEDE, 50 images to be used for retraining. As anticipated in Section 3.3, we retained 5% of the images in the original simulator-based training set, and all the real-world images used to fine-tune the DNN.

In the case of HUDD, we applied its selection algorithm to sample, from an improvement set, 50 images for each RCC (HUDD selects the images that are closer to the RCC centroid). We generated an improvement set with random face images (i.e., generated by randomly selecting simulator parameter values). To avoid bias, we selected for each case study DNN the same number of images as that generated by SEDE (i.e., 50 images for each RCC), which led to a total of 450 images for FLD, 350 for HPD-F, and 500 for HPD-H.

Concerning the random baseline, for each case study DNN, we generated a retraining set consisting of a number of randomly generated simulator images and real-world images. To avoid unfair comparisons, the randomly generated images match in number the images selected by SEDE.

Further, for all the three approaches described above (i.e., SEDE, HUDD, and random baseline), we followed the SEDE retraining process, which consists of retraining the DNN for multiple times and selecting the DNN yielding the best test set accuracy. In our experiments, since each retraining task takes approximately three hours, we performed three retraining tasks since they can be performed overnight, which is acceptable in practice. For all the approaches, we constructed the retraining data set in the same way as SEDE. It consists of the images selected by the approach under analysis, a random selection of images from the original simulator-based training set (5% of the whole set), and all the real-world images used to fine-tune the DNN. To account for randomness, for each approach, we repeated the experiment ten times (i.e., for ten times, we selected the best DNN generated out of three runs). For each approach, we generate ten DNNs.

To positively answer our research questions, SEDE should lead to retrained DNNs with an accuracy that is significantly higher than the one observed when retraining the DNN using either HUDD or the random baseline.

4.6.2 Results. Columns five to seven (i.e., RBL, HUDD, and SEDE) in Table 20 show the average accuracy of the three approaches across ten runs along with the delta with respect to the original DNN model. Also, for SEDE, we report the delta with respect to the best competing approach (i.e., HUDD or random baseline). Finally, we report p-values using a non-parametric Mann–Whitney U-test, for statistical significance, and the $\hat{A}_{12}$ statistics, for effect size.

SEDE, on average, improves the DNN for FLD, HPD-F and HPD-H by +6.07%, +4.50% and +18.65%; the other two approaches, instead, do not improve the HPD-F and FLD DNNs but decrease their performance. For HPD-H, HUDD and the random baseline led to an improvement that is lower than SEDE's, +9.62% and +4.54%, respectively. We believe that in the case of HPD-F and FLD, the retraining based on HUDD and random decrease the DNN performance because the simulator being used generates images that are less realistic; such characteristic leads to the generation of images that are unlikely to include hazard-triggering events and, in turn, the retraining set selected by HUDD and random is not likely to include unsafe images. Since the retraining is performed using (1) the images selected by the approach under analysis, (2) a subset of the original training set, and (3) the real-world training set, safe cases will be over-represented in the training set. In general, unsafe images that are present in the original training set may not be retained for retraining.

Table 20. RQ5: Unsafe set size and the accuracy improvement of SEDE compared to HUDD and random baseline (RBL)

DNN	Original Model Accuracy	Retraining Set Size		RBL Accuracy (Gain)	HUDD Accuracy (Gain)	SEDE		Stat. Sig.	
		Training Set (Sim. + Real)	Unsafe Set			Accuracy (Gain)	Gain over best baseline	p-value	$\hat{A}_{12}$
FLD	80.06%	150 + 6,000	450	77.41% (-2.65%)	79.94% (-0.11%)	86.14% (+6.07%)	**+6.19%**	1.84e-04	1.0
HPD-F	51.65%	1,075 + 476	350	44.33% (-7.32%)	45.80% (-5.85%)	56.15% (+4.50%)	**+10.35%**	4.43e-04	0.94
HPD-H	51.03%	900 + 476	500	55.57% (+4.54%)	60.65% (+9.62%)	69.68% (+18.65%)	**+9.03%**	1.12e-04	1.0

Similar to the above, we believe that, since it is complicated for IEE-Faces to generate unsafe images, in the case of HPD-F and FLD, SEDE led to a more limited improvement in accuracy than in the case of HPD-H: +4.50% and +6.07% vs +18.65%.

Finally, SEDE, on average, improves the DNN results by 9 percentage points over the best competing approach (i.e., +6.19% for FLD, +10.35% for HPD-F, and +9.03% for HPD-H). The difference is always statistically significant (p-value ≤ 0.05) with a large effect size.

Figure 15 shows boxplots with the accuracy of the DNNs retrained using SEDE, HUDD, and the random baseline. Each data point captures the accuracy obtained in one of the ten retraining runs. The boxplots of SEDE overlap with the boxplot of a competing approach only in one case (i.e., the max value for the random baseline), thus suggesting that SEDE performs significantly better with a strong effect size; indeed, competing approaches never lead to a better DNN than SEDE.

4.7 Threats to validity

4.7.1 Internal validity. In our work, we rely on clusters to capture different root causes of a DNN failure where the quality of clusters largely affects the characterization of an unsafe space. This threat is mitigated by empirical results demonstrating that HUDD clusters include images with similar characteristics [17].

4.7.2 External validity. The selection of the case study DNNs and simulators may affect the generalizability of results. We alleviate this issue by selecting subject DNNs that implement classification and regressions tasks motivated by IEE business needs and addressing problems that are quite common in the automotive industry.

Moreover, we rely on two simulators that differ in their fidelity and number of configuration parameters. As we have seen, these characteristics affect the effectiveness of retraining and SEDE's capacity to identify hazard-triggering events. We focus on DNNs processing human faces because they are the focus of our project partners at IEE; we did not consider DNNs performing other tasks (e.g., processing road images) because an industrial case study on that topic was not available for our project.

4.7.3 Conclusion validity. To avoid violating parametric assumptions in our statistical analysis, we rely on a non-parametric test and effect size measure (i.e., Mann Whitney U-test and the Vargha and Delaney's $\hat{A}_{12}$ statistics, respectively) to evaluate the statistical and practical significance of differences in results. When comparing classification results (i.e., RQ4) we applied the Fisher's exact test, which is commonly used in similar contexts.
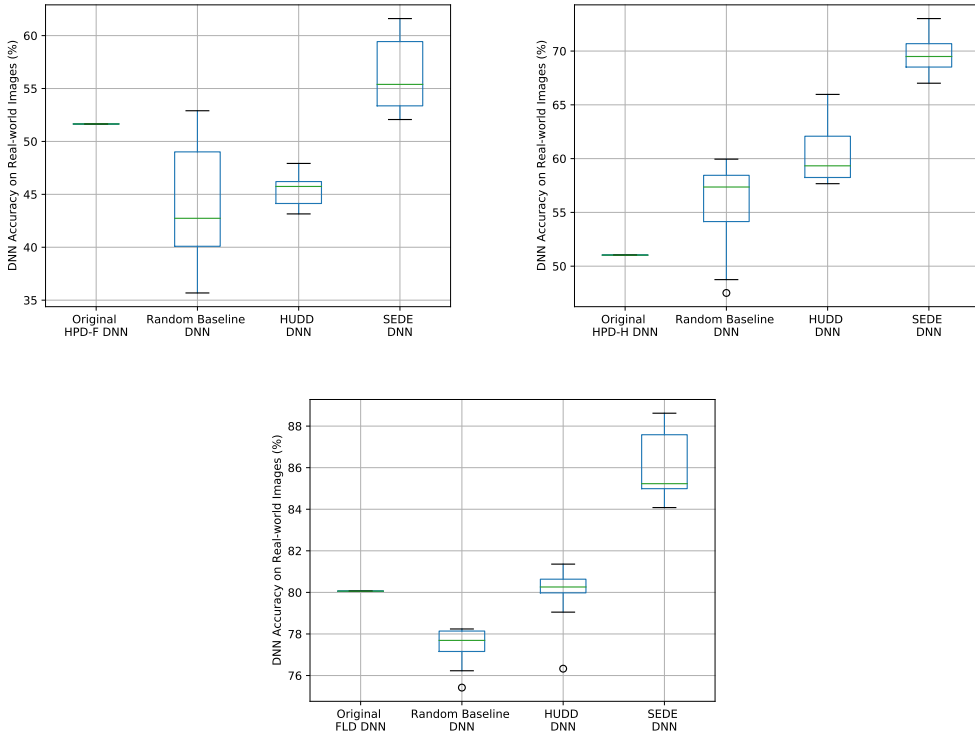
Fig. 15. DNN accuracy on real-world test set images after retraining by SEDE compared to HUDD and random baseline for HPD-F, HPD-H and FLD case study DNNs

Further, for RQ1, we executed the competing approaches for four runs, for each of the 31 RCCs under analysis, to account for randomness and eliminate bias. For RQ2 to RQ4, we compared the results obtained with a large number of images. For RQ2, we considered 25 SEDE images for each RCC and 5, 470 unsafe test set images. For RQ3, we considered 650 SEDE images and 1, 500 randomly generated images. For RQ4, we considered 500 SEDE images for each RCC and 1, 500 random test set images. For RQ5, because of the stochastic nature of SEDE (e.g., DNN retraining), the experiments were executed over thirty runs.

4.7.4 Construct validity. In RQ1 we aim to evaluate if our approach achieves a higher diversity than a state-of-the-art approach. We rely on the chromosome distance computed for each pair of images generated for a RCC, which is based on the parameter values used to generate images with a simulator. By relying on simulator parameter values we can objectively measure diversity.

In RQ2 we evaluate if the images generated by SEDE are close to the medoid of the RCC under analysis. We rely on the heatmap distance since it is the distance metric adopted to generate RCCs.

For RQ3, since simulator parameters drive the selection of specific image characteristics (e.g., head direction), we rely on the variance reduction rate for parameter values to objectively assess the similarity across images belonging to a RCC; variance reduction has also been used to evaluate HUDD [17].

RQ4 evaluates if SEDE expressions are useful for delimiting an unsafe space. RQ5 evaluates the ability of SEDE to improve a DNN. For both, we relied on the accuracy metric (i.e., the percentage of correctly classified images), which is also suggested by safety standards as a mean to evaluate if DNNs can be used for safety-related tasks [32].

4.8   Data Availability

Our implementation of SEDE, the IEE simulators, and the data generated to address our research questions are available online [3, 4]. We cannot share the real-world images used for FLD experiments because they were collected by IEE and protected by privacy agreements.

5   RELATED WORK

INNvestigate [30] and TorchRay [63] are well known tools supporting DNN explanation. However, for explanations concerning DNNs that process images, they generate one heatmap for every failure-inducing input image; each heatmap must be visually inspected by engineers, which makes the investigation of many DNN failures highly expensive. The cost of the manual inspection of heatmaps is one of the problems addressed by HUDD [17], which groups together similar images thus simplifying the identification of the root causes of DNN failures; however, HUDD still requires domain experts capable of visually spotting the commonalities across images, which is no longer needed with SEDE.

Similarly to HUDD, MODE automatically identifies the images to be used to retrain a DNN [42]. However, it cannot identify the root causes of DNN failures; further, in contrast to SEDE, it cannot automatically generate the images to be used for retraining. Like HUDD, MODE's effectiveness remains limited by the availability of an improvement set including unsafe images. Finally, a reusable tool implementing MODE is not available.

DeepJanus characterizes the frontier of DNN misbehaviours by identifying pairs of inputs that are close to each other, with one input leading to a correct DNN output and the other to a DNN failure [58]. It relies on the popular NSGA-II algorithm extended with an archive (to keep the best individuals found in the search) and with repopulation (to escape from stagnation by replacing the most dominated individuals with random ones). Like SEDE, DeepJanus relies on a fitness function that includes a measure of sparseness of the solutions; sparseness is measured as the distance from the closest input in the archive, which is populated with inputs having a distance above a given threshold. Different from DeepJanus, SEDE does not require the configuration of a threshold value, which might be particularly expensive in our context (see Section 2.4). Also, SEDE provides explicit explanations for DNN failures represented using expressions constraining parameter values; DeepJanus, instead, presents example images to end-users thus requiring the visual inspection of images like HUDD. Finally, DeepJanus cannot relate failures observed with images generated by a simulator to failures observed with real-world data, which is a key contribution of SEDE.

Anchors are if-then rules that constrain a subset of the input features so that changes to the unconstrained features do not influence the output of the model to be explained [56]. The Anchors algorithm constructs an explanation rule iteratively, by interacting with the model. At each iteration it alters the values associated to one input feature, till it identifies a range within which the accuracy is above a given threshold. The Anchors algorithm works with the test dataset, not simulators; also, when applied to DNN processing images, it does not generate expressions but identifies the image chunks influencing the DNN output, similarly to heatmap-based approaches.

Kim et al. [34] process images generated with simulators and rely on rule extraction algorithms to characterize correctly and incorrectly classified images in terms of simulator parameters. Unlike SEDE, the approach of Kim et al. cannot be applied to real-world images, which are instead necessary to ensure the applicability of the DNN in the field.

Other works [15, 16, 21, 35, 65, 66] concern DNN retraining approaches that aim to improve the DNN robustness by relying on either image transformations (e.g., rotations) or adversarial inputs. SEDE focuses on DNN accuracy not robustness; also, instead of relying on image transformations, it is the first approach relying on simulators to improve the DNN accuracy observed when the DNN is applied to real-world images.

Some DNN testing approaches can provide explanations for portions of the input space in which DNN failures are observed [2, 26, 27, 69]. For instance, Abdessalem et al. [2] rely on evolutionary algorithms to search for test inputs using simulators and, to maximize test effectiveness, decision trees are used during the search process to learn the regions of the input space that are likely unsafe and, hence, should be targeted by testing. Finally, engineers are presented with decision tree leaves that characterize such portions. Further, recent work studies the effectiveness of decision trees in characterizing the input space of the simulator-based testing process [26, 27]. Finally, DeepHyperion [69] configures a generative model using a metaheuristic search algorithm directed towards generating test inputs in a specific dimension of the inputs space and provides a set of feature maps which visualize the degree of accuracy obtained for different values of dimensions pairs. Different from SEDE, these DNN testing approaches can only be used to characterize simulated scenarios; indeed, they do not integrate any solution (e.g., fitness function based on heatmaps) allowing them to generate inputs that are similar to real-world scenarios.

To summarize, SEDE is the first approach to automatically derive expressions that characterize the unsafe portion of the input space, based on the outputs obtained with real-world images leading to DNN failures. Also, it is the first approach that leverages the generated explanations in order to retrain and improve the DNN.

6   CONCLUSION

The identification of hazard-triggering events is a safety engineering practice that enables engineers to evaluate the risk associated to potentially hazardous behaviors of a system. In this paper, we address the problem of characterizing hazard-triggering events affecting the correct execution of DNN-based systems. We introduced SEDE, a novel approach based on evolutionary algorithms, which generates expressions that characterize hazard-triggering events observed in real-world images processed by DNNs. Such expressions constrain the configuration parameters of a simulator capable of generating images similar to the real-world images under analysis. In turn, they characterize the unsafe portions of the input space.

To identify the unsafe portions of the input space from real-world images, SEDE relies on a state-of-the-art approach, HUDD, that generates clusters (i.e., root cause clusters, RCCs) containing images sharing a common set of characteristics. Such commonalities capture the hazard-triggering events to be investigated by engineers. For each RCC, SEDE performs three distinct executions of evolutionary algorithms; as a result, it generates two sets of images belonging to the RCC, one leading to DNN failures while the other leading to correct DNN outputs. The two sets are then processed by PART, a rule extraction algorithm, to automatically derive SEDE expressions. The evolutionary algorithms employed by SEDE are a modified version of NSGA-II and PaiR, an algorithm introduced in this paper to efficiently generate, using a simulator, images that belong to a RCC and are diverse.

Additionally, SEDE improves the DNN under analysis by retraining it with images that match the generated expressions.

Empirical results conducted with representative case studies in the automotive domain show that (a) our genetic algorithm, PaiR, can generate images for a larger number (87.1%) of RCCs, it generates a larger number of images for each RCC, and the generated images have a significantly higher diversity than those generated by NSGA-II (for all case studies) and DeepNSGA-II (for one

case study), (b) the evolutionary searches employed by SEDE lead to images belonging to the RCCs and having similar characteristics, (c) the expressions generated by SEDE successfully characterize the unsafe input space (i.e., they lead to images showing a DNN accuracy decreased by at least 30% points, on average), and, (d) the retraining process employed by SEDE increases the DNN accuracy up to 18 percentage points with a gain over the best baseline of at least 6 percentage points.

ACKNOWLEDGMENTS

REFERENCES

[1] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing Vision-based Control Systems Using Learnable Evolutionary Algorithms. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. ACM, New York, NY, USA, 1016–1026. https://doi.org/10.1145/3180155.3180160

[2] Raja Ben Abdessalem, Shiva Nejati, Lionel C Briand, and Thomas Stifter. 2018. Testing vision-based control systems using learnable evolutionary algorithms. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 1016–1026.

[3] Authors of this paper. 2022. SEDE code repository. https://github.com/SNTSVV/SEDE

[4] Authors of this paper. 2022. SEDE: replicability package. https://doi.org/10.6084/m9.figshare.19467401.v1

[5] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2016. Testing advanced driver assistance systems using multi-objective search and neural networks. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 63–74.

[6] Jordan J. Bird, Diego R. Faria, Anikó Ekárt, and Pedro P. S. Ayrosa. 2020. From Simulation to Reality: CNN Transfer Learning for Scene Classification. In *2020 IEEE 10th International Conference on Intelligent Systems (IS)*. 619–625. https://doi.org/10.1109/IS48319.2020.9199968

[7] Blender. 2020. Blender 3D simulation and rendering engine. https://www.blender.org/

[8] N. Carlini and D. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 39–57. https://doi.org/10.1109/SP.2017.49

[9] William W. Cohen. 1995. Fast Effective Rule Induction. In *Machine Learning Proceedings 1995*, Armand Prieditis and Stuart Russell (Eds.). Morgan Kaufmann, San Francisco (CA), 115–123. https://doi.org/10.1016/B978-1-55860-377-6.50023-2

[10] MakeHuman community. 2020. MakeHuman computer graphics middleware for the prototyping of humanoids. . http://www.makehumancommunity.org

[11] Piotr Dabkowski and Yarin Gal. 2017. Real Time Image Saliency for Black Box Classifiers. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS?17)*. Curran Associates Inc., Red Hook, NY, USA, 6970–6979.

[12] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197. https://doi.org/10.1109/4235.996017

[13] Steve Dias Da Cruz, Bertram Taetz, Thomas Stifter, and Didier Stricker. 2022. Autoencoder Attractors for Uncertainty Estimation. In *Proceedings of the IEEE International Conference on Pattern Recognition (ICPR)*.

[14] Steve Dias Da Cruz, Bertram Taetz, Thomas Stifter, and Didier Stricker. 2022. Autoencoder for Synthetic to Real Generalization: From Simple to More Complex Scenes. In *Proceedings of the IEEE International Conference on Pattern Recognition (ICPR)*.

[15] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. 2019. Exploring the Landscape of Spatial Robustness. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLRG, Long Beach, California, USA, 1802–1811. http://proceedings.mlr.press/v97/engstrom19a.html

[16] Hasan Ferit Eniser, Simos Gerasimou, and Alper Sen. 2019. DeepFault: Fault Localization for Deep Neural Networks. In *Fundamental Approaches to Software Engineering*, Reiner Hähnle and Wil van der Aalst (Eds.). Springer International

Publishing, Cham, 171–191.

[17] Hazem Fahmy, Fabrizio Pastore, Mojtaba Bagherzadeh, and Lionel Briand. 2021. Supporting Deep Neural Network Safety Analysis and Retraining Through Heatmap-Based Unsupervised Learning. *IEEE Transactions on Reliability* (2021), 1–17. https://doi.org/10.1109/TR.2021.3074750

[18] Gabriele Fanelli, Matthias Dantone, Juergen Gall, Andrea Fossati, and Luc Van Gool. 2013. Random Forests for Real Time 3D Face Analysis. *Int. J. Comput. Vision* 101, 3 (February 2013), 437–458.

[19] Eibe Frank and Ian H. Witten. 1998. Generating Accurate Rule Sets Without Global Optimization. In *Fifteenth International Conference on Machine Learning*, J. Shavlik (Ed.). Morgan Kaufmann, 144–151.

[20] Gordon Fraser and Andrea Arcuri. 2013. Whole Test Suite Generation. *IEEE Transactions on Software Engineering* 39, 2 (2013), 276–291. https://doi.org/10.1109/TSE.2012.14

[21] Xiang Gao, Ripon K. Saha, Mukul R. Prasad, and Abhik Roychoudhury. 2020. Fuzz Testing based Data Augmentation to Improve Robustness of Deep Neural Networks. In *Proceedings of the 42nd International Conference on Software Engineering (ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 10.

[22] Rafael Garcia, Alexandru C. Telea, Bruno Castro da Silva, Jim Torresen, and Joao Luiz Dihl Comba. 2018. A task-and-technique centered survey on visual analytics for deep learning model engineering. *Computers and Graphics* 77 (2018), 30 – 49. https://doi.org/10.1016/j.cag.2018.09.018

[23] FaceShift GmbH. U.S. Patent 9378576, May 2016. Online modeling for real-time facial animation.

[24] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (2015), 1–11.

[25] Fitash Ul Haq, Donghwan Shin, and Lionel Briand. 2022. Efficient Online Testing for DNN-Enabled Systems using Surrogate-Assisted and Many-Objective Optimization. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. 811–822. https://doi.org/10.1145/3510003.3510188

[26] Fitash Ul Haq, Donghwan Shin, Lionel C. Briand, Thomas Stifter, and Jun Wang. 2021. Automatic Test Suite Generation for Key-Points Detection DNNs Using Many-Objective Search (Experience Paper). In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2021)*. Association for Computing Machinery, New York, NY, USA, 91–102. https://doi.org/10.1145/3460319.3464802

[27] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel Briand. 2021. Can Offline Testing of Deep Neural Networks Replace Their Online Testing? A Case Study of Automated Driving Systems. *Empirical Softw. Engg.* 26, 5 (sep 2021), 30. https://doi.org/10.1007/s10664-021-09982-4

[28] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel C. Briand. 2020. Comparing Offline and Online Testing of Deep Neural Networks: An Autonomous Car Case Study. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. 85–95. https://doi.org/10.1109/ICST46399.2020.00019

[29] IEE. 2020. IEE Sensing solution. www.iee.lu.

[30] Innvestigate. 2020. DNN Explanation. https://github.com/albermax/innvestigate

[31] Tadanobu Inoue, Subhajit Choudhury, Giovanni De Magistris, and Sakyasingha Dasgupta. 2018. Transfer Learning from Synthetic to Real Images Using Variational Autoencoders for Precise Position Detection. In *2018 25th IEEE International Conference on Image Processing (ICIP)*. 2725–2729. https://doi.org/10.1109/ICIP.2018.8451064

[32] International Organization for Standardization. 2020. ISO/PAS 21448:2019, Road vehicles: Safety of the intended functionality.

[33] Hisao Ishibuchi, Yuji Sakane, Noritaka Tsukamoto, and Yusuke Nojima. 2009. Evolutionary many-objective optimization by NSGA-II and MOEA/D with large populations. In *2009 IEEE International Conference on Systems, Man and Cybernetics*. 1758–1763. https://doi.org/10.1109/ICSMC.2009.5346628

[34] Edward Kim, Divya Gopinath, Corina Păsăreanu, and Sanjit A. Seshia. 2020. A Programmatic and Semantic Approach to Explaining and Debugging Neural Network Based Object Detectors. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 11125–11134. https://doi.org/10.1109/CVPR42600.2020.01114

[35] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding Deep Learning System Testing Using Surprise Adequacy. In *Proceedings of the 41st International Conference on Software Engineering (ICSE '19)*. IEEE Press, 1039–1049. https://doi.org/10.1109/ICSE.2019.00108

[36] Jiman Kim and Chanjong Park. 2017. End-To-End Ego Lane Estimation Based on Sequential Transfer Learning for Self-Driving Cars. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 1194–1202. https://doi.org/10.1109/CVPRW.2017.158

[37] Ronald S. King. 2014. *Cluster Analysis and Data Mining: An Introduction.* Mercury Learning & Information, USA.

[38] Barbara Kitchenham, Lech Madeyski, David Budgen, Jacky Keung, Pearl Brereton, Stuart Charters, Shirley Gibbs, and Amnart Pohthong. 2017. Robust Statistical Methods for Empirical Software Engineering. *Empirical Software Engineering* 22, 2 (2017), 579–630. https://doi.org/10.1007/s10664-016-9437-5

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* 60, 6 (May 2017), 84–90. https://doi.org/10.1145/3065386

[40] Bingdong Li, Jinlong Li, Ke Tang, and Xin Yao. 2015. Many-Objective Evolutionary Algorithms: A Survey. *ACM Comput. Surv.* 48, 1, Article 13 (sep 2015), 35 pages. https://doi.org/10.1145/2792984

[41] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. 2021. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. *IEEE Transactions on Neural Networks and Learning Systems* (2021), 1–21. https://doi.org/10.1109/TNNLS.2021.3084827

[42] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. 2018. MODE: Automated Neural Network Model Debugging via State Differential Analysis and Input Selection. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018)*. ACM, New York, NY, USA, 175–186. https://doi.org/10.1145/3236024.3236082

[43] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*. 0–28.

[44] ManuelBastioniLAB. 2016. Character creation tool for Blender. https://github.com/animate1978/MB-Lab

[45] Microsoft. 2010. https://developer.microsoft.com/en-us/windows/kinect/.

[46] Christoph Molnar. 2011. *Interpretable Machine Learning*. LuLu. 318 pages. https://christophm.github.io/interpretable-ml-book/

[47] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus Robert Müller. 2019. *Layer-Wise Relevance Propagation: An Overview*. Springer International Publishing, Cham, 193–209. https://doi.org/10.1007/978-3-030-28954-6_10

[48] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus Robert Müller. 2017. Explaining nonlinear classification decisions with deep Taylor decomposition. *Pattern Recognition* 65 (2017), 211 – 222. https://doi.org/10.1016/j.patcog.2016.11.008

[49] Mohamed El Mostadi, Hélène Waeselynck, and Jean-Marc Gabriel. 2022. Virtual Test Scenarios for ADAS: Distance to Real Scenarios Matters!. In *2022 IEEE Intelligent Vehicles Symposium (IV)*. 836–841. https://doi.org/10.1109/IV51971.2022.9827170

[50] Rizwan Ali Naqvi, Muhammad Arsalan, Ganbayar Batchuluun, Hyo Sik Yoon, and Kang Ryoung Park. 2018. Deep Learning-Based Gaze Detection System for Automobile Drivers Using a NIR Camera Sensor. *Sensors* 18, 2 (2018). https://doi.org/10.3390/s18020456

[51] Alejandro Newell, Kaiyu Yang, and Jia Deng. 2016. Stacked Hourglass Networks for Human Pose Estimation. In *Computer Vision – ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Cham, 483–499.

[52] NVIDIA Corporation. 2019. NVIDIA Introduces DRIVE AutoPilot. https://nvidianews.nvidia.com/news/nvidia-introduces-drive-autopilot-worlds-first-commercially-available-level-2+-automated-driving-system

[53] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2018. Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets. *IEEE Transactions on Software Engineering* 44, 2 (2018), 122–158. https://doi.org/10.1109/TSE.2017.2663435

[54] Vitali Petsiuk, Abir Das, and Kate Saenko. 2018. RISE: Randomized Input Sampling for Explanation of Black-box Models. In *Proceedings of the British Machine Vision Conference (BMVC)*.

[55] PyTorch. 2020. PyTorch DNN framework. https://pytorch.org

[56] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Anchors: High-Precision Model-Agnostic Explanations. In *Proceedings of the '18 AAAI Conference on Artificial Intelligence*. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16982

[57] Vincenzo Riccio, Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. 2021. 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 Press, 355–367. https://doi.org/10.1109/ASE51524.2021.9678764

[58] Vincenzo Riccio and Paolo Tonella. 2020. Model-based exploration of the frontier of behaviours for deep learning system testing. *ESEC/FSE 2020 - Proceedings of the 28th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering* 2 (2020), 876–888. https://doi.org/10.1145/3368089.3409730 arXiv:2007.02787

[59] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In *2017 IEEE International Conference on Computer Vision (ICCV)*. 618–626. https://doi.org/10.1109/ICCV.2017.74

[60] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. 2015. Striving for Simplicity: The All Convolutional Net. In *ICLR (workshop track)*. http://lmb.informatik.uni-freiburg.de/Publications/2015/DB15a

[61] DLIB Team. [n.d.].

[62] Tesla, Inc. 2019. "Overview of neural net for vision, sonar and radar processing software". https://www.tesla.com/BLOG/ALL-TESLA-CARS-BEING-PRODUCED-NOW-HAVE-FULL/-SELF-DRIVING-HARDWARE?redirect=no

[63] TorchRay. 2020. DNN Explanation. https://github.com/facebookresearch/TorchRay

[64] Vedat Toğan and Ayşe T. Daloğlu. 2008. An improved genetic algorithm with initial population strategy and self-adaptive member grouping. *Computers & Structures* 86, 11 (2008), 1204–1218. https://doi.org/10.1016/j.compstruc.2007.11.006

[65] Jingyi Wang, Jialuo Chen, Youcheng Sun, Xingjun Ma, Dongxia Wang, Jun Sun, and Peng Cheng. 2021. RobOT: Robustness-Oriented Testing for Deep Learning Systems. In *Proceedings of the 43rd International Conference on Software Engineering (ICSE '21)*. IEEE Press, 300–311. https://doi.org/10.1109/ICSE43902.2021.00038

[66] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. Deephunter: A coverage-guided fuzz testing framework for deep neural networks. *ISSTA 2019 - Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis* (2019), 158–168. https://doi.org/10.1145/3293882.3330579

[67] Matthew D. Zeiler and Rob Fergus. 2014. Visualizing and Understanding Convolutional Networks. In *Computer Vision – ECCV 2014*, David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars (Eds.). Springer International Publishing, Cham, 818–833.

[68] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. 2016. Learning Deep Features for Discriminative Localization. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2921–2929. https://doi.org/10.1109/CVPR.2016.319

[69] Tahereh Zohdinasab, Vincenzo Riccio, Alessio Gambi, and Paolo Tonella. 2021. Deephyperion: exploring the feature space of deep learning-based systems through illumination search. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 79–90.