Focused Test Generation for Autonomous Driving Systems

TAHEREH ZOHDINASAB, Universita della Svizzera italiana, Lugano, Switzerland
VINCENZO RICCIO, University of Udine, Udine, Italy
PAOLO TONELLA, Universita della Svizzera Italiana, Lugano, Switzerland

Testing Autonomous Driving Systems (ADSs) is crucial to ensure their reliability when navigating complex environments. ADSs may exhibit unexpected behaviours when presented, during operation, with driving scenarios containing features inadequately represented in the training dataset. To address this shift from development to operation, developers must acquire new data with the newly observed features. This data can be then utilised to fine tune the ADS, so as to reach the desired level of reliability in performing driving tasks. However, the resource-intensive nature of testing ADSs requires efficient methodologies for generating targeted and diverse tests.

In this work, we introduce a novel approach, DeepAtash-LR, that incorporates a surrogate model into the focused test generation process. This integration significantly improves focused testing effectiveness and applicability in resource-intensive scenarios. Experimental results show that the integration of the surrogate model is fundamental to the success of DeepAtash-LR. Our approach was able to generate an average of up to 60× more targeted, failure-inducing inputs compared to the baseline approach. Moreover, the inputs generated by DeepAtash-LR were useful to significantly improve the quality of the original ADS through fine tuning.

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

Additional Key Words and Phrases: Software testing, deep learning, search based software engineering, autonomous driving systems

ACM Reference Format:
Tahereh Zohdinasab, Vincenzo Riccio, and Paolo Tonella. 2024. Focused Test Generation for Autonomous Driving Systems. *ACM Trans. Softw. Eng. Methodol.* 33, 6, Article 152 (June 2024), 32 pages. https://doi.org/10.1145/3664605

1 INTRODUCTION

Deep Learning (DL) is progressively emerging as a fundamental component of complex software systems, including **Autonomous Driving Systems (ADSs)**. DL systems are highly significant in the field of **Software Engineering (SE)**, due to their capability to acquire knowledge for complex tasks through training data [35]. This capability can also have negative implications, as DL

This work was partially supported by the H2020 project PRECRIME, funded under the ERC Advanced Grant 2017 Program (ERC Grant Agreement n. 787703), and by the project Toposcope (SNF grant n. 214989).
Authors' Contact Information: Tahereh Zohdinasab (Corresponding author), informatics, Universita della Svizzera italiana, Lugano, Ticino, Switzerland; e-mail: tahereh.zohdinasab@usi.ch; Vincenzo Riccio,University of Udine, Udine, Italy; e-mail: vincenzo.riccio@uniud.it; Paolo Tonella, Universita della Svizzera Italiana, Lugano, Switzerland; e-mail: paolo.tonella@usi.ch.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM 1049-331X/2024/06-ART152
https://doi.org/10.1145/3664605

systems may exhibit unexpected behaviours when presented with inputs containing features that are absent or inadequately represented in the training dataset [27]. For instance, a **lane-keeping assist system (LKAS)** might predict an erroneous steering angle when presented with a road with an exceptionally sharp turn, if this particular variation is not adequately represented in the training set. The consequences of a misbehaviour might be catastrophic for life-critical systems like ADSs, potentially endangering all road participants (e.g., driver, passengers, and pedestrians). Hence, ADSs necessitate rigorous testing employing appropriate techniques capable of generating data beyond the datasets used during development [45, 55].

Test generation approaches specific to ADSs are designed to automatically produce misbehaviour-inducing driving scenarios [2, 10–12, 19, 23, 26, 29, 46]. However, when developers encounter inputs that result in a misbehaviour in the field, it becomes crucial for them to understand the underlying causes of misbehaviour. For instance, they must identify which input features are underrepresented and responsible for the observed misbehaviour [43, 56]. Consequently, developers need to obtain or generate new data that includes such features to effectively address the issue. Ben Abdessalem et al. [2] use ranges of input/environment variables (e.g., pedestrian position/speed) to automatically identify the most critical regions of the input space. However, their approach has limitations, as input/environment variables do not fully characterise higher level abstract properties of the road as well as behavioral properties of the driving system. Moreover, in their approach the user cannot specify a specific range of interest, as the approach automatically focuses on input values that most likely trigger failures. SAMOTA [23] introduces a surrogate model to make the test generation process for ADS more efficient and correspondingly more effective at exposing misbehaviours. Whether such efficiency boost translates into an increased capability of focusing the test generation process on interesting and possibly unexplored combinations of driving scenario features remains an open research question, which we address in our empirical study.

Feature maps [60] automatically organize and characterise misbehaviour-inducing inputs based on human interpretable features encompassing both structural and behavioural aspects of the considered inputs. A feature map provides a representation of the feature space, which is defined by a set of N relevant dimensions of variation (i.e., the map axes, each corresponding to an input feature). In a feature map, test inputs are placed based on their feature values. These maps can provide insights into a test set, such as revealing feature value combinations associated with tests that triggered misbehaviours or indicating the likelihood of observing a misbehaviour for each feature combination. Figure 1 shows an instance of a bi-dimensional feature map for a LKAS. This feature map is defined by the car's mean lateral position and the maximum road curvature. Each combination of feature values corresponds to a map cell and is shaded to represent the likelihood of experiencing a misbehaviour, with darker colors indicating higher probabilities. In the example of Figure 1, the lane-keeping system under test is likely to fail for roads with very sharp turns where the car drives close to the road margins. Feature maps have been applied in prior **Software Engineering (SE)** research, serving various purposes, such as test generation [60, 61], test selection [41], misbehaviour explanation [65], and test adequacy assessment [10, 11].

During the testing phase, feature maps identify the regions within the feature space that lack sufficient coverage [10]. However, during operation, it is possible to encounter critical feature values that are under-represented in the train/test datasets used during development for which new and diverse driving scenarios need to be collected and manually labelled [22]. Therefore, testers need to find multiple misbehaviour-inducing test inputs associated with specific feature combinations. These additional inputs can be used to improve the quality of the ADSs in production, by fine tuning them on such new data.

DeepAtash [62] was the pioneer focused input generator for DL systems targeting human-interpretable features, as it introduced a novel approach to generate misbehaviour-inducing inputs

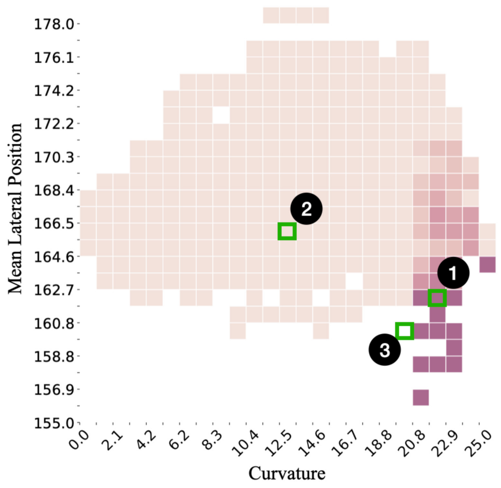

Fig. 1. Feature map for a lane keeping assistant system. The axes quantify mean lateral position and curvature of the roads. The cells report the probability of exposing a misbehaviour for the corresponding feature value combinations, i.e., darker colors correspond to higher misbehaviour probabilities.

with specific, user-defined feature values. This tool was successfully used for different usage scenarios, such as to collect new diverse inputs with critical, misbehaviour-inducing characteristics (❶ in Figure 1), to stress the system to expose failures with inputs that do not seem critical (❷), or to generate new data with underrepresented or unseen feature values (❸). For instance, a DL system deployed in operation could encounter feature combinations that were rarely (❶, ❷) or never (❸) observed during its development phase.

Through its search-based focused testing approach, DeepAtash proved its effectiveness by generating multiple diverse inputs with predefined target feature values for different DL systems, i.e., image and text classifiers. However, DeepAtash may not be the ideal choice for testing ADSs. In fact, its evolutionary nature demands multiple system executions to produce misbehaviour-inducing inputs within the specified target. Testing ADSs is known to be resource-intensive, since it requires expensive executions (e.g., simulations) to assess the system behaviour. Consequently, DeepAtash might invest most of its test generation time-budget exploring unpromising regions of the feature space, i.e., those that do not contain the target features or have a minimal likelihood of causing misbehaviours, because each execution (i.e., simulation in the candidate test scenario) consumes a substantial fraction of the available budget.

In this work, we propose a novel focused test generator named DeepAtash-LR, specifically designed for ADSs, with the primary goal of overcoming the aforementioned limitations of its predecessor. The core innovation of DeepAtash-LR is its ability to reduce the computational resource demands associated with the evaluation of less promising solutions. Our approach achieves this goal by integrating a surrogate model within the evolutionary process. Surrogate models have been extensively used for ADS testing since they closely emulate system behaviour, while drastically reducing computational overhead [40]. Within DeepAtash-LR, the surrogate model plays a pivotal role in predicting the likelihood of a misbehaviour and the behavioural features of the generated inputs. Consequently, DeepAtash-LR executes the system only when the input is likely to belong to the target feature map cell and is likely to trigger a misbehaviour. This strategic approach optimizes the utilisation of computational resources in ADS testing and makes a dramatically better use of the available test generation budget.

Our evaluation of DeepAtash-LR was conducted in the context of a LKAS DL component, using the BeamNG driving simulator [8] across different usage scenarios. Our empirical results show that the surrogate model is indispensable for generating misbehaviour-inducing inputs within the predefined targets. Moreover, the inputs generated by DeepAtash-LR have been used to fine tune the ADS under study and enhance its performance on under-represented feature combinations, which were initially not handled at all (i.e., on failure-inducing feature combinations). After fine tuning, the system improved remarkably, with no significant regressions.

In comparison to the original paper describing DeepAtash, the main extensions that can be found in this paper are:

— an input generation approach for ADS focused on features that do not represent just values of input/environment variables, but rather represent higher-level structural/behavioral properties of the test scenarios. This approach is an extension to ADSs of our focused test generation approach, initially applied only to image and text classifiers;
— DeepAtash-LR, a novel tool that efficiently performs focused test generation by leveraging a surrogate model. To the best of our knowledge, our solution is the first to integrate feature-driven, focused testing and surrogate models in the ADS domain;
— a large empirical study which shows that DeepAtash-LR outperforms DeepAtash and another state-of-the-art test generator, DeepHyperion-CS, in generating focused tests. In particular, we empirically show that focused test generation is feasible in the ADS domain only if a surrogate model is integrated into the search algorithm;
— an experiment showcasing the practical application of DeepAtash-LR to an ADS development task, i.e., fine tuning.

These extensions collectively contribute to the advancement and applicability of focused test generation to ADSs.

To encourage open research, we release the code of DeepAtash-LR and the experimental data at:
https://github.com/testingautomated-usi/deepatash

2 BACKGROUND ON FEATURE MAPS

A *feature map* represents the space of the features that are relevant for characterising the test inputs. Such features can either be *structural,* i.e., characterising the input itself (e.g., the road curvature) or *behavioural,* i.e., characterising the system's output when exercised by the given input (e.g., the mean lateral position of the car when driving on the input road). In particular, we refer to the high-level, human-interpretable input features defined by experts in the work by Zohdinasab et al. [60], such as the mean lateral position of the car and the maximum curvature of the road, as shown in Figure 1. Each test input is placed within the corresponding cell of the feature map. This assignment is achieved by computing metrics that quantify each input feature. The map granularity (i.e., the number of map cells in each dimension) is configurable and can be changed depending on the desired level of discrimination. The level of granularity can influence the discriminative capability of the corresponding map, i.e., its ability to separate misbehaviours from correct behaviours. If the granularity is too low, it may group together too many inputs, leading to an inadequate characterization of misbehaviors. As recommended in the original paper [61], users can empirically define the granularity for each feature as the ratio between the desired granularity, i.e., the desired number of cells, and the expected range of the feature. Typically, opting for a reasonably high granularity (e.g., at least 25×25 cells) has proven sufficient for ensuring effective discrimination and characterisation.

Multiple inputs belonging to the same map cells, allow the analysis of the probability of triggering misbehaviours in the corresponding feature space region. *Misbehaviour probability maps* are feature maps that report, for each cell, the **Average Misbehaviour Probability (AMP)** observed in different test suites. AMP is computed as the ratio of the number of misbehaviour-inducing inputs to the total number of inputs in each cell.

In the misbehaviour probability map shown in Figure 1, the cells' shade is proportional to their AMP values (i.e., darker cells correspond to higher AMP values), while blank cells correspond to feature combination values not covered by the available test inputs.

Feature maps have been used in the literature for different testing tasks. DeepHyperion-CS [60, 61] is a test generator that leverages feature maps for exploring the feature space at large and finding failure-inducing inputs with different features. Nguyen et al. [41] used feature maps for test selection; they ensure test diversity by selecting inputs that occupy different map cells. Feature maps can also assess test suite adequacy, measured as the number of feature cells covered by the test inputs in the test suite. A recent search-based testing competition adopted feature maps to compare different test generators [10, 18], while the study by Biagiola et al. [11] compares the feature maps generated by the same test generator on different driving simulators.

In this work, we rely on feature maps to identify the target feature values that interesting test scenarios should cover and to guide DeepAtash-LR's focused test generation process. Although we considered pairwise combinations of features (i.e., bi-dimensional maps) to facilitate visualisation and discussion of the results, our approach can work also with higher-dimensional maps.

3 FOCUSED TEST GENERATION USING SURROGATE MODEL

The primary objective of DeepAtash-LR is to generate test inputs with user-specified characteristics (i.e., feature combinations) that trigger misbehaviours of the system under test. To achieve this goal, DeepAtash-LR leverages the feature maps introduced in Section 2. Specifically, it generates inputs falling within a predefined feature map cell that trigger unexpected behaviours of the system, while striving to maximize the sparseness among the generated solutions to produce a wide range of diverse inputs.

Building upon our prior research [62], we adopt an evolutionary search based approach to achieve our goal. This generates inputs which optimise three fitness functions, i.e., they (1) are close to the target cell; (2) trigger a misbehaviour of the ADS; and (3) are diverse from the already found solutions. Given the desired target ranges of feature values, referred to as the *target cell* (e.g., $[1:5] \times [10:15]$ if we want the first feature f_1 to be between 1 and 5 and the second f_2 between 10 and 15), DeepAtash-LR guides the generation of novel inputs towards the feature subspace defined by the specified values. DeepAtash-LR follows an iterative process wherein it takes an initial set of inputs (referred to as *seeds*) and proceeds to manipulate them repeatedly until they eventually fall within or close to the target cell defined by the user. The evolutionary process within DeepAtash-LR is guided by fitness functions representing the distance to the target cell, the closeness to misbehaviour and the distance from the previously discovered solutions.

The original DeepAtash algorithm [62] computes the closeness to misbehaviour of the generated inputs by running the ADS in the generated driving scenarios within a simulator, which can be quite computationally expensive. It should be noticed that also the computation of behavioural features requires the execution of the ADS under test in a simulator. This translates into multiple executions of the ADS under test in each iteration. Even though evolutionary search algorithms generally demonstrate good scalability, their ability to effectively generate inputs with specific features may decrease when the evaluation of such inputs is expensive. A clear example of this occurs when testing the lane-keeping assist component of an ADS. In this context, the assessment of the system's behaviour in each input scenario requires the execution of time-intensive simulations, which eventually constrain the evolutionary process by restricting the number of iterations performed within a given time budget (i.e., the number of generations of the evolutionary search). In this work, we address this limitation by proposing a way to improve the efficiency of the search-based algorithm, which translates also into its effectiveness, as a better use of the available time budget might lead to higher fault detection. Specifically, we employ surrogate models to predict the system's behaviour without the need for executing time-consuming simulations and, thus, saving time during the optimization process. Surrogate models have recently gained popularity [1, 13, 21, 23, 40] for emulating the behaviour of the original system. Surrogate models are trained on datasets containing inputs and the corresponding behaviours of the systems. They approximate and mimic the behaviour of the original system without executing it, hence providing a faster and more efficient way to evaluate the fitness of candidate test inputs, although such evaluation is affected by some degree of approximation, due to the use of a surrogate instead of the real system. Hence, their practical usefulness depends on the degree of approximation involved and on the tolerance to approximation errors of the test generation algorithm, which can be only assessed empirically. With surrogate models, ADS simulations are executed only when necessary, such as when collecting data to train the surrogate model or when storing an input in the archive as a final solution. The surrogate model allows the algorithm to decide whether an input is close enough to the target or exhibits the desired characteristics without the need to run costly simulations for all inputs.

Algorithm 1 presents an overview of our new focused test input generation technique. This general pseudocode can be instantiated either as a single- or multi-objective optimization process. The highlighted lines of pseudocode indicate the changes over the original DeepAtash algorithm. The algorithm starts by initialising an empty archive A (line 1), that will store the best test inputs generated during the search, i.e., the most sparse inputs with feature values falling within or close to the target ranges. Then, the algorithm proceeds by generating an initial population P (function InitialisePopulation at line 2), consisting of a specified number of individuals. These individuals are instantiated by selecting elements from an initial pool of seeds S, which is provided as input to the algorithm. Typically, S can be a subset of the test set available in the data set of the ADS under test. Indeed, if a test set is not available or desirable, the subset S can alternatively be composed of randomly generated inputs to serve as the initial pool for the test input generation process. The first phase is concluded by determining the fitness values of all the individuals of the initial population without using the surrogate model (lines 3-4). Since there is no data available about the behaviour of the original ADS at this stage, we need to prepare the training data set for the surrogate model based on these initial evaluations. Then, the algorithm initialises *TrainingDS*, i.e., the training data set for the surrogate model, with the inputs from the current population P (line 5).

The main evolutionary loop is performed until the termination condition is met (lines 6–22). During the evolutionary loop, training of the surrogate model occurs only when a sufficient number of iterations has been performed and, thus, enough inputs have been generated to train the surrogate model. Once this condition is satisfied (line 7), the algorithm trains the surrogate model (line 8), and sets the *UseSurrogate* variable to *True* (line 9). At each iteration, the population is mutated by genetic operators to produce its offspring Q (lines 11–14). The Repopulation operator avoids stagnation in local optima by replacing the worst individuals of the population (i.e., the ones with the lowest fitness values) with new inputs selected from the initial pool of seeds S (line 15). In particular, Repopulation takes as input the archive A to avoid selecting seeds already used to generate individuals present in the current archive A.

Function Evaluate calculates the fitness of the current population P and its offspring Q. This function avoids redundant evaluations of the distance from the target cell and the closeness to

ALGORITHM 1: DeepAtash-LR's Focused Test Generation.

```
Input: $B$: execution budget
       $archivesize$: target archive size
       $S$: set of input seeds
       $popsize$: population size
       $targetCell$: target of focused test generation
       $epsilon$: threshold for surrogate training
Output: $A$: archive of test inputs in the target cell
1  $A \leftarrow \emptyset$;
2  $population\ P \leftarrow$ InitialisePopulation($S$, $popsize$);
3  $UseSurrogate \leftarrow False$ ;
4  Evaluate($P$, $A$, $targetCell$, $UseSurrogate$) ;
5  $TrainingDS \leftarrow$ P ;
6  while $elapsedBudget < B$ do
7      if $elapsedBudget > epsilon \times B$ and $UseSurrogate$ is $False$ then
8          TrainSurrogate($TrainingDS$) ;
9          $UseSurrogate \leftarrow True$ ;
10     end
11     $offspring\ Q \leftarrow P$ ;
12     foreach $q \in Q$ do
13         $q \leftarrow$ Mutate($q$) ;
14     end
       // substitute the worst individuals
15     $P \leftarrow$ Repopulation($P$, $S$, $A$);
16     Evaluate( $P \cup Q$, $A$, $targetCell$, $UseSurrogate$) ;
17     if $UseSurrogate$ is $False$ then
18         $TrainingDS \leftarrow TrainingDS \cup Q$ ;
19     end
20     $A \leftarrow$ UpdateArchive($P \cup Q$, $archiveSize$, $targetCell$, $UseSurrogate$);
21     $P \leftarrow$ Select($P \cup Q$, $popsize$);
22 end
23 $A \leftarrow$ FilterMisbehaviours($A$);
24 return $A$
```

misbehaviour for the individuals that have been already evaluated in the previous iterations, i.e., the individuals from P that have not undergone repopulation. However, the evaluation of sparseness is performed at each iteration as it measures how dissimilar a new individual is from individuals previously discovered and stored in the archive. In fact, the sparseness value of an individual may vary at each iteration based on the current composition of the archive. The evaluation of the behaviour and the feature values can be performed either using the surrogate model or the system under test depending on the value of the *UseSurrogate* variable (line 16). The evaluated inputs are continuously added to the training data set of the surrogate model only when we have used the actual ADS for their evaluations (line 17-19).

The inputs that are close to the target cell, i.e., those with a distance from the target cell smaller than a certain threshold, are saved in the archive A if they outperform the previously found

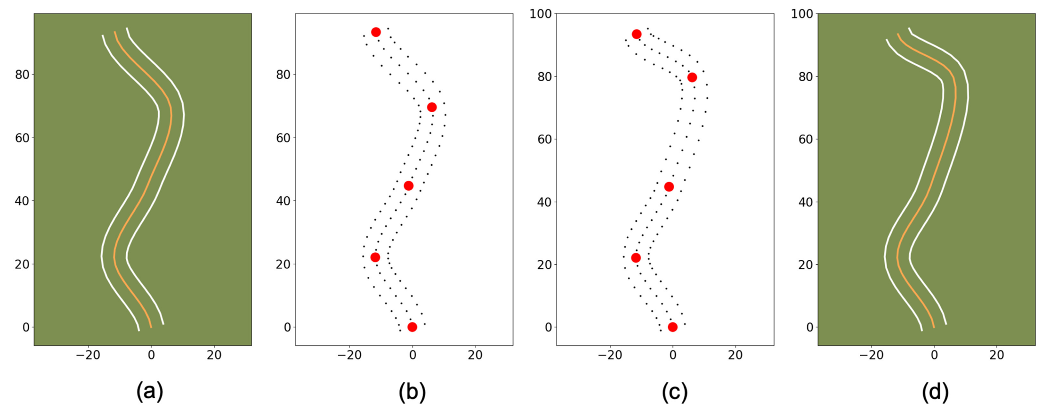

Fig. 2. Road input representation and mutation. (a) original input; (b) original input model; (c) model mutated by moving a control point; (d) mutated input.

solutions. In other words, if these inputs exhibit better fitness than the existing stored solutions, they are added to the archive (line 20). It should be noticed that when an archive replacement is supposed to happen, because a better individual was generated, the actual fitness values are needed. If the individual's evaluation was approximated by the surrogate model, when the individual becomes a candidate for the archive it must be first evaluated against the real system, which means the ADS must be simulated in the candidate input scenario.

Then, the SELECT function selects the *popsize* fittest individuals for the next generation (line 21). When dealing with the optimization of multiple fitness functions simultaneously, the ranking of individuals for selection in DEEPATASH-LR is determined based on the principles of Pareto dominance and crowding distance, as prescribed by the NSGA-II multi-objective optimization algorithm [16] (see Section 3.5). When the execution budget B is finished, the algorithm filters the misbehaviour-inducing inputs from the archive and reports them as final outcome (lines 23-24), together with the AMP values of each feature map cell.

In the rest of this section, we provide a detailed description of the key aspects of DEEPATASH-LR and how we applied it to the lane-keeping application domain.

3.1 Input Representation

DEEPATASH-LR can be classified as a model-based input generation technique [57], since it represents individuals, i.e., test inputs, as model instances and directly manipulates the model to generate new data. Model-based techniques are widely adopted in various domains, including safety-critical areas such as the automotive industry [34]. Model-based test input generation has been already successfully applied to the ADS domain [1–3, 44, 46, 60–62], where a generative model is available, in particular, for the lane-keeping task.

In our setting, test inputs are scenarios in which the car drives within the BEAMNG simulator. Following the state-of-the-art [19, 46, 60, 61], we represent a driving scenario to test the lane keeping component as a plain asphalt road surrounded by green grass on which the car has to drive by keeping the right lane. Such simulated roads are modelled as a sequence of consecutive points in a bi-dimensional space, interpolated by Catmull-Rom cubic splines [15]. DEEPATASH-LR uses the recursive algorithm for the evaluation of Catmull-Rom cubic splines proposed by Barry and Goldman [7] to transform a model's instance (i.e., Catmull-Rom's control parameters) into a road that can be rendered in the simulator. Figure 2(a) and (b) show an original road input and its corresponding input model, respectively. In the input model, the larger red dots denote the control points of the Catmull-Rom spline representing the central line, while the smaller grey dots represent the interpolated points defining the road shape.

3.2 Fitness Functions

DeepAtash-LR's optimization process utilises three fitness functions. They quantify: (1) the distance of the test input from the target cell; (2) the closeness of the ADS to cause a misbehaviour when executing the given test input; and, (3) the distance of the input from the already found solutions (i.e., its sparseness).

3.2.1 Distance from the Target Cell. To quantify the distance of an individual x from the target cell c, DeepAtash-LR employs the Manhattan distance calculation. The algorithm computes the Manhattan distance between the cell containing the individual x and the target cell c. The objective is to minimize this fitness function, since the algorithm aims to find individuals that are closer to the target cell.

$$\min \mathit{fitness}_1(x) = \min \mathit{dist}(x, c) \tag{1}$$

Given a target cell $c = [l_1 : u_1] \times \ldots \times [l_N : u_N]$, with N the number of features being defined, the range size $s_i = u_i - l_i$ along each dimension f_i (with $i \in \{1, \ldots, N\}$) determines the Manhattan distance of a given individual x from the target cell c, according to the following equations:

$$d(x_i, c_i) = \begin{cases} \left\lceil \frac{l_i - x.f_i}{s_i} \right\rceil, & \text{if } x.f_i < l_i \\ 0, & \text{if } l_i \leq x.f_i < u_i \\ \left\lceil \frac{x.f_i - u_i}{s_i} \right\rceil, & \text{if } x.f_i > u_i \\ 1, & \text{if } x.f_i = u_i \end{cases} \tag{2}$$

$$d(x, c) = \sum_{i=1}^{N} d(x_i, c_i) \tag{3}$$

Along each dimension i, the difference between the individual's coordinate $x.f_i$ and the cell's lower/upper bound (l_i or u_i), divided by the cell size s_i, gives the number of cells that separate x and c along f_i (the value is rounded up, to get an integer). To ensure that the target cells remain distinct from other cells, we defined the lower bound to be within the target cell and the upper bound to be outside of it. Therefore, the Manhattan distance is set to 1 when the feature value $x.f_i$ is equal to the upper bound of a target cell because we consider the individual x_i as belonging to the adjacent cell. Instead, the distance is set to 0 when $x.f_i$ is equal to the lower bound of a target cell because we regard the individual x_i as belonging to the target cell. This design choice prevents overlaps between adjacent cells. The sum of the number of separating cells across all dimensions corresponds to the Manhattan distance between x and c. Let us consider as an instance a target cell $c = [3:6] \times [4:8]$ and a candidate solution x with feature values $x.f_1 = 9, x.f_2 = 5$. The Manhattan distance between x and c is $\lceil (9-6)/3 \rceil + 0 = 1 + 0 = 1$.

3.2.2 Closeness to Misbehaviour. DeepAtash-LR aims to generate test inputs that trigger misbehaviours of the ADS under test. Therefore, it employs a problem-specific fitness function that quantifies how close the ADS is to misbehaving when exercised with the evaluated input. By minimizing this fitness function, DeepAtash-LR identifies inputs that are more likely to trigger misbehaviours.

$$\min\ \mathit{fitness}_2(x) = \min\ \mathit{evaluateBehaviour}(x) \tag{4}$$

For the *lane-keeping problem*, we characterise the behaviour of the system by the maximum distance of the car from the center of the right lane during the simulation [30, 46, 53]. The fitness value is calculated as $min(w/2-d)$, where w is the lane width and d the distance of the car from the center of the right lane. This fitness function gets its maximum value $(w/2)$ when the car is at the center of the right lane, whereas it gets a negative value when there is a misbehaviour, i.e., the car is beyond one of the lane margins. All instances where this fitness function has a negative value indicate a misbehaviour, i.e., the vehicle does not keep the lane. To ensure equal importance for all misbehaviours, we cap the value of this fitness function for all misbehaviour-inducing individuals to a small negative value (i.e., -0.1). To evaluate this fitness function, DeepAtash-LR can use either the original ADS under test or the surrogate model trained with the inputs generated during the evolution, along with the corresponding behaviours observed through simulation.

3.2.3 Sparseness. Ensuring diversity and distinctiveness in the generated test inputs is essential for the effectiveness of a focused test input generator. By generating a wide variety of inputs that are significantly different from one another, the generator can cover a broader range of execution scenarios, which helps in thoroughly exploring the system's behaviour.

To achieve this objective, DeepAtash-LR uses a fitness function that quantifies how dissimilar an input is from the solutions previously discovered during the search. By maximizing this fitness function, DeepAtash-LR encourages the generation of novel and unique inputs, leading to a richer and more comprehensive set of test cases that can reveal different aspects of the system's behaviour. More precisely, DeepAtash-LR calculates the sparseness of the individual x with respect to the individuals in the archive A as follows:

$$\max\ fitness_3(x) = \max\ sparse(x, A) \tag{5}$$

Function *sparse* computes the minimum distance of x from the solutions in the archive A: $\min_{y \in A, y \neq x} \text{dist}(x, y)$. The distance function *dist* is computed on pairs of inputs and is domain-specific. Specifically, we use the weighted Levenshtein distance [46]. This metric considers the minimum number of edit operations to transform one road into the other. Edit operations apply to the two sequences of angles sampled on the spines of the two roads being compared. This distance metric is suitable for the comparison of shapes of roads with different lengths and is robust against translation and rotation. In fact, it takes into account the order of the points along the road spines, as well as the relative angle between consecutive points.

3.3 Surrogate Model

As shown in Algorithm 1, each iteration of DeepAtash-LR involves the evaluation of newly generated individuals to compute their behavioural features and fitness values. This evaluation can be performed using either the original ADS or the surrogate model, depending on the value of the *UseSurrogate* variable. The surrogate model functions as a black-box component, accepting test input (e.g., sequence of control points which represents the road shape) as input and generating the desired variable (e.g., a behavioural feature) as output. DeepAtash-LR requires a set of labeled inputs to train an accurate surrogate model. Consequently, a portion of the time budget is dedicated to evaluating the generated inputs using the original ADS under test through simulations. Once a reasonable amount of training data has been accumulated (i.e., after a given number of iterations), this data is utilized to train the surrogate model. The amount of collected training data is determined using the threshold ϵ in Algorithm 1, which defines the portion of time budget dedicated to training data collection. This threshold can be chosen through preliminary runs to ensure that the performance of the surrogate model is satisfactory. It is essential to set the threshold ϵ at a relatively low value; otherwise, adopting the surrogate model would be inefficient as the training data collection time and the training time would be excessively long.

The trained surrogate model becomes a valuable asset for DeepAtash-LR within the remaining time budget. Instead of relying on the original ADS for evaluations, our tool can use the surrogate model to predict the behaviour of the ADS for new inputs (line 9). This replacement may reduce the time and computational resources needed for evaluations, if the surrogate model is capable of delivering fast and accurate predictions. For problems such as lane-keeping, where input evaluation involves costly simulations, utilizing the surrogate model can significantly impact the exploration of the algorithm.

We adopted three distinct surrogate models to predict two behavioural features (i.e., mean lateral position and standard deviation of steering angle) along with the closeness to misbehaviour (i.e., distance to the road boundaries). These predictions would need the execution of simulations in the original configuration of DeepAtash.

By incorporating the surrogate model, DeepAtash-LR enables the evolutionary search to explore the feature space more effectively and efficiently. The surrogate model reduces the computational burden by providing faster evaluations, facilitating a more thorough exploration of the search space in high-cost domains, such as the automotive one.

3.4 Archive of Solutions

The archive of solutions stores the best individuals encountered throughout the search process. Upon reaching the last iteration of the search, the archive contains the final solutions, i.e., the inputs within or close to the target cell. The archive is particularly useful to prevent the *cycling* phenomenon, i.e., when the search moves from one cell of the feature space to another and back again, with no memory of the cells it has already explored [39]. The archive helps to maintain diversity in the search by preserving effective solutions, thus ensuring a more comprehensive exploration of the feature space and avoiding redundant iterations.

The UpdateArchive function that manages the fixed-size archive of solutions is described in Algorithm 2. In DeepAtash-LR, the surrogate model can be employed to predict the behaviour of the original ADS, allowing for faster fitness evaluations during the evolutionary process. However, when an individual is deemed as eligible for inclusion in the archive by the surrogate model, i.e., its predicted distance from the target cell is lower or equal to 1, DeepAtash-LR re-evaluates it by performing a simulation using the original ADS. As shown in lines 3-4 of Algorithm 2, the re-evaluation takes place only if the candidate was evaluated through the surrogate model. Such re-evaluation recomputes the values of the behavioural features and of the fitness functions, allowing to obtain a more accurate and reliable assessment. In fact, this additional evaluation step ensures that individuals that are close enough to the target cell undergo a final assessment with the original ADS to confirm their suitability for inclusion in the archive. By doing so, DeepAtash-LR maintains the integrity of the archived solutions and ensures that the best-performing inputs are validated against the real ADS system.

When the archive is not yet full (i.e., it contains less solutions than its predefined target size), DeepAtash-LR adopts an inclusive approach towards candidate individuals. All individuals that reach the target cell or the neighboring feature map cells are included in the archive (lines 6-8). Otherwise, if the archive reaches its maximum capacity, the new candidate input competes with the worst individual currently present in the archive, i.e., the individual stored in the archive with the highest distance to the target and (for equal distances to the target) the lowest sparseness (line 9). The competition is based on the values of $fitness_1$, $fitness_2$, and $fitness_3$ of both the new candidate input and the worst individual in the archive (lines 10-26). If the candidate individual has a lower distance to the target than the worst archived individual, UpdateArchive replaces the

ALGORITHM 2: The UpdateArchive function.

```
Input:  P: population
        A: initial archive
        targetCell: target feature value ranges
        UseSurrogate: if surrogate model is used or not
Output: A: updated archive
1  foreach $p \in P$ do
2      if DIST(p, targetCell) $\leq 1$ then
3          if UseSurrogate then
4              Evaluate(p, A, targetCell, UseSurrogate=False);
5          end
6          if A is not full and $p \notin A$ and DIST( p , targetCell) $\leq 1$ then
7              A.insert(p) ;
8          else
9              ind $\leftarrow$ getWorstIndividual(A);
10             if DIST(p, targetCell) < DIST(ind, targetCell) then
11                 A.insert(p) ;
12                 A.remove(ind) ;
13             else
14                 if DIST(p, targetCell) == DIST(ind, targetCell) then
15                     if p.behaviour < ind.behaviour then
16                         A.insert(p) ;
17                         A.remove(ind) ;
18                     else
19                         if p.behaviour == ind.behaviour & p.sparse > ind.sparse then
20                             A.insert(p) ;
21                             A.remove(ind) ;
22                         end
23                     end
24                 end
25             end
26         end
27     end
28 end
29 return A ;
```

worst individual within the archive with the candidate individual. In other words, the candidate individual takes the place of the worst-performing individual in the archive if it outperforms the latter in terms of distance to the target. This ensures that the archive retains the best-performing individuals, which are closer to the target (lines 10-12). When the compared inputs have equal distance to the target, the algorithm compares their closeness to misbehaviour and keeps in the archive the best one, which is closer to induce a misbehaviour (lines 14-17). If the compared individuals have the same distance to the target and closeness to a misbehaviour, they compete on the value of their sparseness: the sparser input is kept, while the other is discarded (lines 19-22). Eventually (line 29), UpdateArchive returns the updated archive *A*.

At the end of the search process, the FilterMisbehaviours function (line 27 of Algorithm 1) is executed to filter and retain only the misbehaviour-inducing inputs in the archive. Since the archive may also contain correctly-behaving inputs, this filtering step is necessary to focus solely on the inputs that cause misbehaviours in the ADS under test.

3.5 Search Strategies

Selection of the best individuals (line 21 of Algorithm 1) requires a search strategy in the space of the candidate individuals. Similar to our previous work [62], we evaluated two different search strategies for DeepAtash-LR: Single-Objective search, which optimises only the distance to the target, and Multi-Objective search, which explicitly rewards also the closeness to misbehaviour and the sparseness.

3.5.1 Single-Objective Search. As a single-objective search strategy, we opt for a **Genetic Algorithm (GA)** due to its demonstrated effectiveness in test generation [17]. Specifically, we utilise a population-based GA that aims to minimize the Manhattan distance to the target cell. In each iteration, the GA selects the best individuals from the current population and the offspring based on their single fitness value. These selected individuals are then included in the next population, facilitating the evolution of test inputs towards those that are closer to the target cell.

3.5.2 Multi-Objective Search. In the multi-objective strategy, the focused test generation search algorithm optimizes all three fitness functions defined in Section 3.2. To accomplish this, we employ the NSGA-II algorithm [16], which is widely utilized in Software Engineering [2, 33, 36, 42, 44, 46] and has been reported to be highly effective for generating test cases. NSGA-II is well-known for its ability to handle multiple objectives and efficiently explore the search space to find a diverse set of solutions that represent different trade-offs between the multiple objectives. NSGA-II applies Pareto front analysis to identify and prioritize solutions that are not dominated by any other individual, representing the best trade-offs among the fitness functions. Specifically, in NSGA-II, a solution x is considered to dominate another solution y if x is at least as good as y in all fitness values and strictly better than y in at least one fitness value. The ranking of individuals in NSGA-II is based on Pareto dominance, where non-dominated fronts are selected and removed from the solutions one after the other. Within the same Pareto front, individuals are further prioritized based on their crowding distance. The crowding distance ensures diversity by selecting individuals that are farther apart in the objective space, promoting a balanced spread of solutions along the Pareto front [16]. Indeed, the adoption of the NSGA-II search strategy, with its multi-objective optimization approach, introduces additional computational overhead due to the evaluation of multiple fitness functions, Pareto dominance, and crowding distances. However, this strategy offers potential advantages as it explicitly promotes diverse and misbehaviour-inducing inputs.

3.6 Mutation

The Mutate function (line 13 in Algorithm 1) mutates an existing input to generate a new input. This operator applies a perturbation, uniformly sampled in a customisable range, to the input model's control parameters. More specifically, DeepAtash-LR mutates the road geometry by applying a displacement to the coordinates of the model's control points (see Figure 2). Each time an input is mutated, DeepAtash-LR verifies that the mutant complies with the domain-specific validity constraints. In particular, DeepAtash-LR verifies that the mutant is different from its parent and it is a valid road, i.e., (1) the start point and the end point of the road should be different, (2) the road should fall within a square bounding box of fixed size, and (3) a road should not self-intersect. If any of these checks fails, the mutation operator is applied repeatedly, until a valid input is obtained.

3.7 Population Management

DeepAtash-LR starts its search from an initial population of size *popsize*, which is obtained by selecting from a larger pool of inputs, named *seeds*. Function InitialisePopulation (line 2 in Algorithm 1) selects the *popsize* individuals closest to the target from the seeds S. More specifically, to generate the seed pool S, we randomly generate valid roads, i.e., roads with different start/end points that do not self-intersect and are entirely contained within a squared bounding box of fixed size.

In search-based approaches, including DeepAtash-LR, one common challenge is the possibility of getting stuck in local optima, which can limit the exploration of the search space despite the efforts to promote diversity through mechanisms such as our sparseness fitness function (see Equation (5)). To address this issue and foster increased variation within the population, DeepAtash-LR incorporates a Repopulation operator.

At each iteration of Algorithm 1, the Repopulation operator replaces a portion of the worst-performing individuals in the current population, specifically those with the lowest fitness values (line 15). The new individuals are generated from a randomly selected seed that is not already present in the current population or in the archive.

4 EXPERIMENTAL EVALUATION

4.1 Research Questions

The goal of our evaluation is to understand the effectiveness of our approach in generating misbehaviour-inducing test inputs with the desired features. In particular, we compare two alternative surrogate model candidates, assess different alternative configurations of DeepAtash-LR, compare the best DeepAtash-LR configuration with an existing state-of-the-art test generator (DeepHyperion-CS [61]), and investigate the usefulness of the generated test inputs for improving ADSs. Therefore, we answer the following research questions:

RQ0 (DNN vs LR): *Which type of surrogate model is more efficient and effective?*

A crucial aspect of our approach involves the careful selection of the surrogate model, which is tailored to the specific characteristics of the domain. To this aim, we explored the possibility of employing either **Linear Regression (LR)** or **Deep Neural Networks (DNN)** as surrogate models to predict the behaviour of the driving agent. LR fits a linear model with coefficients $(\beta_0, \ldots, \beta_p)$ to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\min_{\beta_0, \beta_1, \ldots, \beta_p} \sum_{i=1}^{n} \left(y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip})\right)^2$$

where n is the number of observations, p is the number of features, y_i is the observed target for the ith observation, and x_{ij} is the jth feature value for the ith observation. We adopted three separate LR models for predicting two behavioural feature values and the fitness value measuring closeness to misbehaviour.

As DNNs are general function approximators even for non linear functions, we considered two distinct DNN architectures. The former consists of three dense layers with *Sigmoid* activation, aimed to predict the closeness to misbehaviour. In fact, *Sigmoid* is a nonlinear function that supports also negative values. The latter consists ofthree dense layers with *ReLU* activation, aimed to predict each behavioural feature. We use *ReLU* activations as this function is monotonic in the positive range, consistently with the input values that range between 0 and infinity. While there are other alternatives for designing the architecture of the surrogate models, we chose a simple 3-layered architecture as ours is not a complex task and because we wanted to keep our approach efficient, i.e., with low training and prediction time. We also wanted to keep the number of parameters small, as the expected training set size is small.

RQ1 (Surrogate Model): *Does the surrogate model improve the effectiveness of the original tool, DeepAtash?*

This RQ assesses whether the use of the surrogate model impacts the effectiveness of focused test generation in DeepAtash-LR. Specifically, our objective is to compare the performance of DeepAtash-LR with the previous version, DeepAtash, which does not utilize surrogate models.

RQ2 (Single vs Multi Objective): *Is DeepAtash-LR mode effective in the single or in the multi objective configuration for generating focused test inputs?*

DeepAtash-LR can be alternatively configured with single- or multi-objective search strategies, as explained in Section 3.5. This RQ aims at comparing the effectiveness of such two alternative configurations.

RQ3 (Comparison): *How does DeepAtash-LR compare with the state-of-the-art tool DeepHyperion-CS?*

In this RQ, we are interested in whether DeepAtash-LR outperforms DeepHyperion-CS [61] in generating test inputs in proximity of and within the target cell. We compare the best performing DeepAtash-LR configuration (obtained from RQ2) against DeepHyperion-CS, as the latter is the only state-of-the-art test generator that targets the feature space at large by means of an illumination search algorithm. DeepHyperion-CS tries to cover all feature combinations and thus it is more likely to produce inputs on the selected target than, e.g., random techniques, which may produce few or no inputs on the target, making the comparison with DeepAtash-LR impossible. To the best of our knowledge, no other DL test generator is *focused*, i.e., capable of targeting a specific region of the feature space. Moreover, DeepHyperion-CS [61] is a model-based test input generator that is applicable to BeamNG and shares the same input representation and mutation genetic operators as DeepAtash-LR. Such similarities allow for a fair and unbiased experimental comparison by eliminating confounding factors, which helps us to assess the specific and isolated contribution of our focused algorithm and the associated surrogate model (DeepAtash-LR) to the test input generation process.

RQ4 (Usefulness): *Can the test inputs generated by DeepAtash-LR be used to improve the ADS system under test?*

To investigate the usefulness of DeepAtash-LR in a common DL usage scenario, we simulate a situation where a dev2op data shift is observed. This means that a particular feature combination is frequently encountered during the operation of the DL system, but it was inadequately represented or not present at all during the system's development phase, e.g., in the original training data. In this context, DeepAtash-LR serves as a valuable tool for testers to address this data shift and target specific feature values of interest. By generating test inputs that focus on underrepresented and critical feature combinations, testers can effectively fine-tune the DL system. In this way, the generated test inputs are used to improve the quality of the DL system. Obviously, testers should assess also that fine-tuning does not introduce regressions or unintended side effects, e.g., the system may learn how to deal with the new test inputs, but might "forget" the correct behaviour for inputs belonging to the original training set.

4.2 Metrics

In this study, we define feature maps by considering high-level features that effectively characterize a self-driving scenario from the input and behavioural viewpoint. Specifically, we resorted to the features proposed in the DeepHyperion-CS paper [61]. These features and the metrics to measure them were obtained by adopting a systematic methodology for feature definition and metric identification [60]. For this work, we chose the following features that cover both structural and behavioral attributes of the test inputs, thereby providing a comprehensive assessment of the driving agent's quality.

— ***Standard Deviation of Steering Angle (StdSA):*** measures the level of activity exhibited by the driving agent on the steering wheel and can be used to quantify passenger comfort and driving quality;
— ***Mean Lateral Position (MLP):*** represents the ego-car's positioning within the right lane. It is computed as the mean distance between the center of the car and the right lane margins;
— ***Turn Count (TurnCnt):*** corresponds to the number of direction changes between consecutive road segments, with a requirement that the angle of change be at least 5°.;
— ***Maximum Curvature (Curv):*** provides insight into the severity of the turns composing each road, by computing the reciprocal of the minimum road turn radius.

To compare alternative surrogate models, we use **Mean Absolute Error (MAE)** and **Mean Squared Error (MSE)**. These metrics are commonly used to evaluate the performance of ADS components in offline mode, i.e., without performing simulations. Given a set of predictions $\hat{y}_i$ and the corresponding expected values y_i for n data points, MAE and MSE are computed as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |(\hat{y}_i - y_i)| \tag{6}$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \tag{7}$$

A lower MAE/MSE indicates that the model's predictions are closer to the expected values, suggesting a better fit of the model to the data. In addition, we report the training time of each model to assess their efficiency.

We evaluate DeepAtash-LR's effectiveness by counting the inputs that fall within or close to the target cell. To this aim, we measure the ***Tests Close to the target (TC)***, a metric that quantifies the number of generated misbehaviours in the proximity of the target feature map's cell, i.e., the solutions stored in the archive with distance to the target lower than or equal to 1. Additionally, we assess DeepAtash-LR's ability to cover the target cell by computing the number of *Tests on Target (TT)*, i.e., the number of misbehaviours that fall exactly within the boundaries of the target cell.

We favor a test input generator that generates diverse inputs for the target feature map cell since it may allow to discover multiple causes of misbehaviours. To evaluate the diversity of test inputs, we resort to the *T**ests Close to the target Diversity (TCD)*** and ***Tests on Target Diversity (TTD)*** metrics. To this aim, we project the generated inputs in a lower dimensional space by using the ***t-distributed Stochastic Neighbor Embedding (t-SNE)*** algorithm [25, 58], which projects similar inputs to neighbouring points and dissimilar inputs to distant points with high probability. Specifically, we project the inputs generated by all approaches being compared onto the same t-SNE space and group neighbouring points using a clustering algorithm in such a space. The diversity value for each approach is computed by counting the number of clusters that contain at least one input generated by the respective approach. These counts are then divided by the total number of clusters. The resulting values represent the input diversity produced by each test generator [13]. For each, test generation approach, TCD and TTD measure the relative cluster coverage obtained by the tests close to the target or on target, respectively. A higher diversity value signifies that the inputs are spread across a broader range of distinct clusters, indicating a more varied and comprehensive exploration of the feature subspace of interest by the test generator.

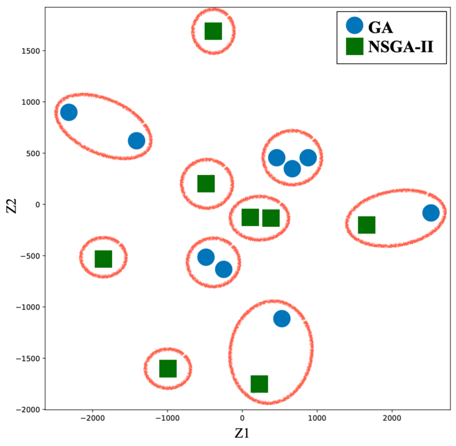

Fig. 3. Example t-SNE plot to explain the computation of *test diversity* metrics, with clusters represented as empty circles containing inputs (smaller, solid shapes).

We configured t-SNE by selecting 2 as the number of dimensions, as it performed well in preliminary runs and facilitated the interpretation of the results. For the t-SNE perplexity, which affects how inputs are scattered or concentrated, we set it to 0.1 based on visual inspection of the plots obtained with different values. To perform clustering, we applied the *k-Means* algorithm [6] and performed Silhouette analysis [48] to determine the optimal number of clusters k^*, i.e., the value with optimal balance between cohesion and separation of the clusters.

Figure 3 exemplifies the diversity comparison between two DeepAtash configurations. The circles and squares represent the inputs generated by different tool configurations in the 2D t-SNE space. The samples are grouped into clusters (represented as circles enclosing samples) and diversity is computed as the number of clusters covered by each configuration. In this example, the diversity value is 0.7(7/10) for NSGA-II (green squares) and it is 0.5(5/10) for GA (blue circles).

We evaluate DeepAtash-LR's usefulness by assessing the driving model's performance after fine tuning it on DeepAtash-LR's inputs. We consider both offline and online evaluation. Offline evaluation involves testing the DL model using pre-collected data without real-time interaction, while online evaluation involves deploying the DL model in a real-time, interactive environment, typically using a simulator.

MAE and MSE are particularly useful for assessing regression tasks, in which the goal is to predict continuous numerical values. We measure MAE and MSE before and after fine tuning the model in offline mode. To evaluate the model in online mode, we measure ***Success Rate*** **(SR)** which indicates the ability of the self-driving car to drive on the road without any failure (i.e., without driving out of the boundaries of road). Online evaluation is conducted to verify that the model successfully passes the regression test scenarios [24, 52].

4.3 Subject System

We evaluate DeepAtash-LR on a popular, safety-critical DL-based ADS. Specifically, we considered the DAVE-2 end-to-end driving agent, which utilizes the DL architecture developed by Bojarski et al. [14]. Such architecture consists of a sequence of three convolutional layers and five fully-connected layers. The objective of DAVE-2 is to address a regression task by predicting steering angles based on images captured by the on-board camera, ensuring the ego-car remains within the right road lane. DAVE-2 has been adopted as subject system by a large number of studies

Fig. 4. Test execution within the BeamNG simulator.

on ADS and DL system testing [11, 28, 31, 32, 46, 50, 51, 54, 60, 61]. We integrated the DAVE-2 model into the BeamNG.tech driving simulator [8] to reproduce the behaviour of the selected ADS within synthetically generated driving scenarios. BeamNG.tech is extensively employed in the software testing literature [55], as it features a soft-body dynamics simulation rooted in a spring-mass framework. Such simulation capability facilitates precise emulation of physical attributes, including vehicle deformation, offering a valuable tool for both researchers and practitioners (see Figure 4).

Since DAVE-2 operates through behavioural cloning, i.e., it learns to establish a mapping between images and steering angles based on examples, we exclusively utilized positive examples for training. Specifically, we trained DAVE-2 with images captured by the BeamNG.tech on-board camera coupled with corresponding steering angles of the ego-car, automatically collected by running the simulator's autopilot on 40 randomly generated roads (the autopilot computes optimal trajectories based on global knowledge of the environment, which is unavailable when the DNN is driving and the only input comes from the camera). The training process consisted of 200 epochs, with batches of size 128 and a learning rate of 0.0001. The good quality of the trained model was evaluated using a test set comprising 10 randomly generated roads, distinct from the ones used in the training. The model adopted for our experiments was able to correctly perform the driving task on all the roads of the test set, achieving a success rate of 100%.

4.4 Evaluation Scenarios

We leveraged misbehaviour probability maps and the AMP metric to replicate various selections of the target cell from the maps reported in Figure 5. This was achieved through the assessment of DeepAtash-LR in the following scenarios:

Dark Targets: targets selected from cells characterized as *dark* (i.e., misbehaviour probability > 0.8). These cells correspond to feature values that are more prone to misbehaviours. In this scenario, the user aims to collect a diverse set of new inputs that possess critical attributes. Such a choice might be driven by the intention to improve the DL system's performance;

Grey Targets: targets corresponding to covered cells with misbehaviour probability ≤ 0.8. These cells represent combinations of feature values that generally lead the DL system to function as intended. Consequently, in this scenario the user wants to stress the DL system with inputs characterized by seemingly non-critical attributes. This approach facilitates an exploration of whether such inputs could potentially trigger misbehaviours, thus contributing to a comprehensive assessment of the DL system's robustness;

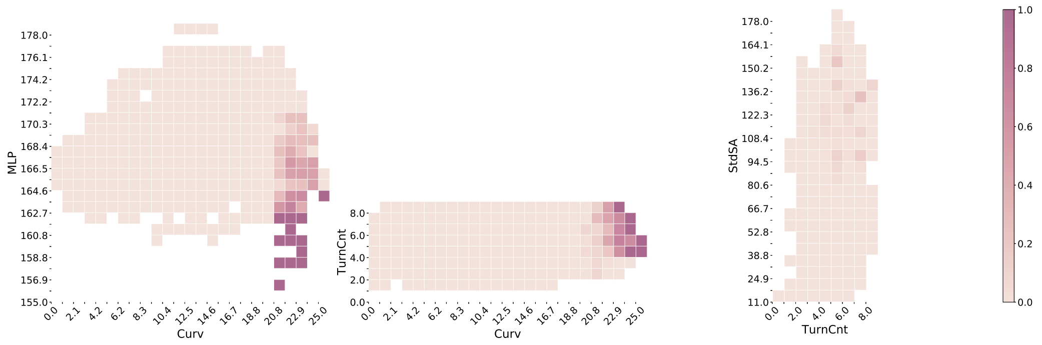

Fig. 5. Average Misbehaviour Probability (AMP) maps generated by DeepHyperion-CS for BeamNG. The axes quantify different features. The cells report the probability of exposing a misbehaviour for the corresponding feature value combinations, i.e., darker colors correspond to higher misbehaviour probabilities.

White Targets: cells that do not contain any input. This scenario reflects gaps within the dataset and, consequently, in the knowledge of the DL system under test. The user's objective is twofold: firstly, to generate new data that covers the missing combinations of feature values; and secondly, to assess the self-driving car's behaviour and potential outcomes for the chosen feature value combinations.

4.5 Experimental Procedure

To answer our research questions, we ran DeepAtash-LR in the three evaluation scenarios outlined in Section 4.4, along with DeepHyperion-CS, on the subject system. We focused our analysis on a subset of three pairs of features, chosen from the total of six possible pairs, due to practical considerations. Specifically, the cost of conducting complex and resource-intensive driving simulations escalates significantly. To ensure a controlled and effective evaluation process, we carefully selected three representative and diverse pairs of features that were likely to provide valuable insights into the performance and behaviour of the DL system. In particular, we chose a combination of two structural features (i.e., *Curv-TurnCnt*) and two combinations consisting of a structural and a behavioural feature (i.e., *TurnCnt-StdSA* and *Curv-MLP*). These selections are particularly interesting because for these combinations the exploration achieved by DeepHyperion-CS is not as extensive as for other feature combinations (which justifies a focused approach) and because they contain a lower number of misbehaviours.

For each evaluation scenario, the first step is the selection of the target cell from the misbehaviour probability maps generated by DeepHyperion-CS (see Figure 5). Specifically, we require that the chosen target cell contains a number of instances that falls below the average count observed across all cells within the feature map. This filtering mechanism ensures that cells with comparatively fewer occurrences are prioritized for selection, tackling the scarcity of data in such regions. Correspondingly, we randomly chose dark cells and grey cells with a coverage (i.e., number of individuals assigned to the cell) achieved by DeepHyperion-CS lower than the average cell coverage across all DeepHyperion-CS's runs.

Notably, this filter is not applicable to white cells, as they do not contain any existing data. For white targets, we selected cells that were not covered in the DeepHyperion-CS misbehaviour probability maps. However, since uncovered cells might correspond to unfeasible feature combinations, we randomly selected white cells that were adjacent (with a distance ≤ 1) to covered cells.

We ran DeepAtash-LR focusing on the identified target cells and collected the resulting archives of solutions. The hyperparameters of DeepAtash-LR were derived, whenever possible, from the

Table 1. Hyperparameters used in the Experiments

Parameter	BeamNG
seed pool size	80
initial pop size	48
population size	10
time budget (s)	36000
repopulation upper bound	2
target archive size	10
number of epochs for retraining	20
learning rate for retraining	0.0001

experiments conducted with DeepHyperion-CS, reported in its original paper [60]. We fine tuned these hyperparameters through a few preliminary runs to ensure that the target cells could be reached. The values of the hyperparameters are presented in Table 1.

We initiated the search process by randomly generating a pool of seeds, the quantity of which was defined as the *seed pool size*. These seeds serve as starting points for the search processes. From this pool of seeds, we selected the *initial pop size* inputs that were closest to the target cell. These chosen inputs constituted the initial population for the targeted test generation. By starting the search in proximity to the target, we focus the optimization process on relevant regions, thereby increasing the likelihood of generating inputs that trigger failures and possess feature values of interest.

To allow statistical analysis, we performed the same number of runs for each configuration of every test generator. We ran DeepAtash-LR 10 times for each type of target (i.e., dark, grey and white targets), for each feature combination (Curv-TurnCnt, TurnCnt-StdSA, Curv-MLP), and for each considered search strategy (i.e., GA and NSGA-II). Since there were no dark target areas in the AMP of the $TurnCnt-StdSA$ feature combination, we excluded the dark target area specifically for this target type – feature combination pair. Hence, in total we performed $10 \times (3 \times 3 - 1) \times 2 = 160$ DeepAtash-LR runs. As for DeepAtash, we started our experimentation considering one feature combination for each type of target. Correspondingly, DeepAtash underwent execution with two search strategies for three pairs of target type and feature combination, resulting in a total of $2 \times 3 \times 10 = 60$ DeepAtash runs. Since after collecting these results the superiority of DeepAtash-LR was undoubtedly obvious, we preferred to save experimentation time and we did not run the remaining five pairs of target type – feature combination. DeepHyperion-CS is not a focused test generator, so it does not require to be executed for different target types. Consequently, we run DeepHyperion-CS 10 times for each of the three feature combinations ($10 \times 3 = 30$ DeepHyperion-CS runs) In summary, the total number of runs performed was 250 (160 DeepHyperion-CS + 60 DeepAtash + 30 DeepAtash-LR).

To ensure a fair comparison, we ran all tools with the same time budget (i.e., 10 hours). To assess the statistical significance of the comparisons between DeepAtash and DeepAtash-LR (RQ1), between different DeepAtash-LR configurations (RQ2), and between DeepAtash and DeepHyperion-CS (RQ3), we applied the Mann-Whitney U-test and measured the effect size by means of the Vargha-Delaney's $\hat{A}_{12}$ statistic [5].

In our experiments we used the best-performing surrogate model found by RQ0. Specifically, we ran DeepAtash-LR on a predefined target cell with a time budget of 2 hours to collect the training data and trained the surrogate models, i.e., a LR and a DNN. Then, we assessed the surrogate models on a random pool of 40 valid roads. To allow statistical analysis, we repeated this experiment 10 times, using a different test set for each repetition.

Table 2. RQ0 - Mean Absolute Error (MAE), Mean Squared Error (MSE) and Training Time (Time) for DNN and Linear Regression (LR) as Surrogate Model for DeepAtash-LR

	MLP			StdSA			Closeness to Misbehaviour		
Surrogate Model	MAE	MSE	Time (s)	MAE	MSE	Time (s)	MAE	MSE	Time (s)
DNN	26.062	2837.120	4.154	31.208	999.487	2.7436	0.211	**0.056**	2.461
LR	**2.076**	**4.823**	**0.059**	**27.834**	**793.185**	**0.000**	**0.308**	0.102	**0.001**

In each column, boldface indicates the minimum for MSE and time; underline indicates values significantly higher than the remaining ones (p-value < 0.05, non-negligible effect size).

To address RQ4, we performed fine tuning [9] of the DL model under test. The fine tuning process involved extending the model's training by conducting additional epochs under the same configuration (see Table 1). Such training was performed using the test inputs generated by DeepAtash-LR that were close to (TC) and within (TT) the designated target regions. To this aim, we collected all the inputs generated by DeepAtash-LR within and close to the targets. Then, we divided these inputs into two distinct sets, i.e., $training_{DA}$ and $test_{DA}$, 80% for training and the remaining 20% for testing. The combination of the original training set and $training_{DA}$ was used to fine tune the DL system. This choice reduces the risk of forgetting the learned task, by ensuring that both original training data and newly generated inputs are simultaneously available during training. As described in Section 4.3, we generated 10 random valid roads to serve as the test set. This test set and $test_{DA}$ were employed to evaluate the accuracy improvement of the fine-tuned DL system and verify if the driving agent exhibited a decline in handling inputs that were previously managed correctly before fine-tuning. To establish the statistical significance of the improvement in accuracy, we repeated the fine-tuning procedure 10 times for every run of DeepAtash-LR on each target cell. This iterative approach allowed us to gather sufficient data and obtain reliable statistical results.

5 RESULTS

5.1 RQ0: DNN vs LR

To choose the most suitable surrogate model for our purpose, we compared the candidate models, i.e., DNNs and LRs (see Section 4). Table 2 shows the results achieved by adopting DNN and LR models in terms of MAE and MSE along with the time needed to train them. LR achieved significantly lower MAE, MSE and training time than DNN for both behavioural features. Remarkably, LR achieved 588× lower MSE than DNN for the MLP feature. As fitness predictors, i.e., for the value of closeness to misbehaviour, DNN led to better predictions in terms of MSE, while MAE and training time were significantly worse than LR. Hence, we ultimately selected LR as the surrogate model for DeepAtash-LR.

The good performance of LR suggests that the relationship between the input features and the fitness values can be assumed to be approximately linear.

Summary: *Linear Regressors are more effective and efficient than DNNs as surrogate models for predicting behavioural features as well as the performance of the considered ADS.*

5.2 RQ1: Surrogate Model

To assess the usefulness of the surrogate model, we conducted a comparative analysis between DeepAtash-LR and DeepAtash. Specifically, we considered only the three evaluation scenarios in which both test generators were focused on the same type of target for the same feature combination (see first two columns of Table 3 and Table 4).

Table 3. RQ1 - Tests Close to Target (TC), Tests on Target (TT), Tests Close to Target Diversity (TCD), and Tests on Target Diversity (TTD) by DeepAtash and DeepAtash-LR

		DeepAtash				DeepAtash-LR			
		GA		NSGA-II		GA		NSGA-II	
	Features	TC [TCD]	TT [TTD]	TC [TCD]	TT [TTD]	TC [TCD]	TT [TTD]	TC [TCD]	TT [TTD]
Dark	Curv-TurnCnt	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]	**4.30** [**0.50**]	**3.70** [**0.40**]	**7.40** [**0.90**]	**6.40** [**0.80**]
Grey	TurnCnt-StdSA	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]	**2.40** [**0.70**]	**2.30** [**0.70**]	**3.70** [**0.70**]	**1.70** [**0.50**]
White	Curv-MLP	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]	**4.80** [**0.50**]	**2.90** [**0.30**]	**4.70** [**0.50**]	**4.70** [**0.50**]

In each row, boldface indicates the maximum metric values between DeepAtash and DeepAtash-LR, both with GA (resp. NSGA-II); underline indicates values significantly higher than the remaining ones (p-value < 0.05, non-negligible effect size).

Table 4. RQ1 - Number of Iterations Performed by DeepAtash and DeepAtash-LR in the Same Time Budget

		DeepAtash		DeepAtash-LR	
	Features	GA	NSGA-II	GA	NSGA-II
Dark	Curv-TurnCnt	116.10	132.70	**187.30**	**183.10**
Grey	TurnCnt-StdSA	104.70	122.70	**155.20**	**196.80**
White	Curv-MLP	117.60	114.40	**312.80**	**284.80**

In each row, boldface indicates the maximum metric values between DeepAtash and DeepAtash-LR, both with GA (resp. NSGA-II); underline indicates values significantly higher than the remaining ones (p-value < 0.05, non-negligible effect size).

Table 3 presents the results achieved by DeepAtash and DeepAtash-LR, using two distinct search strategies (GA and NSGA-II). As illustrated in the table, DeepAtash was not able to generate inputs close to the target, when operating without the support of a surrogate model. In fact, DeepAtash achieved zero test inputs both directly on the target (TT) and in close proximity to the target (TC) across all of its runs. Conversely, by integrating a Linear regression model within the search process, DeepAtash-LR exhibited a substantial enhancement. This improvement is manifested in the presence of diverse inputs generated, both in close proximity to the target (TC) and directly on the target (TT), across all feature combinations.

Furthermore, Table 4 reports the number of iterations performed by DeepAtash and DeepAtash-LR within the given time budget. This count of iterations serves as an indicator of the extent of the search process undertaken by each approach during the test input generation procedure. DeepAtash-LR executed a significantly higher number of search iterations compared to DeepAtash, for all targets (i.e., up to 199 more iterations for the white target). This increase can be attributed to the time-saving advantage gained by leveraging the surrogate model to avoid expensive simulations.

These results indeed affirm the crucial role of the surrogate model in directing DeepAtash-LR towards the target feature space through increased (surrogate) evaluations, thereby resulting in the creation of more focused test inputs, reaching the target.

Summary: *The integration of surrogate models into the focused test generation process is beneficial in scenarios where evaluations entail resource-intensive simulations. The adoption of surrogate models allowed DeepAtash-LR to navigate the feature space with increased efficiency and efficacy, facilitating the production of diverse misbehaviour-inducing inputs in proximity of and within the target cells. In our case study, without surrogate model DeepAtash did not reach any target cell and performed substantially less search iterations than DeepAtash-LR.*

Table 5. RQ2 - Tests Close to Target (TC), Tests on Target (TT), Tests Close to Target Diversity (TCD), and Tests on Target Diversity (TTD) by Alternative DeepAtash-LR Configurations

		GA		NSGA-II	
	Features	TC [TCD]	TT [TTD]	TC [TCD]	TT [TTD]
Dark	Curv-MLP	5.50 [0.50]	**5.50** [**0.44**]	**6.70** [**0.63**]	5.30 [0.34]
	Curv-TurnCnt	4.30 [0.36]	3.70 [0.31]	**7.40** [**0.65**]	**6.40** [**0.65**]
Grey	Curv-MLP	1.60 [0.19]	1.50 [0.12]	**5.80** [**0.71**]	**4.60** [**0.47**]
	Curv-TurnCnt	2.10 [0.31]	0.50 [0.10]	**4.10** [**0.51**]	**0.70** [0.10]
	TurnCnt-StdSA	2.40 [0.50]	**2.30** [**0.30**]	**3.70** [**0.54**]	1.70 [0.15]
White	Curv-MLP	**4.80** [0.36]	2.90 [0.26]	4.70 [0.36]	**4.70** [**0.44**]
	Curv-TurnCnt	4.10 [0.38]	3.30 [0.30]	**7.80** [**0.74**]	**7.20** [**0.63**]
	TurnCnt-StdSA	0.00 [0.00]	0.00 [0.00]	**2.00** [**0.50**]	0.00 [0.00]
	AVG	3.10 [0.32]	2.46 [0.22]	**5.27** [**0.58**]	**3.82** [**0.34**]

In each row, boldface indicates the maximum of each of the four metrics; underline indicates values significantly higher than the remaining ones (p-value < 0.05, non-negligible effect size).

5.3 RQ2: Single vs Multi-objective

Table 5 reports the results achieved by the two configurations of DeepAtash-LR, which adopt alternative search strategies. Specifically, we implemented GA as single-objective approach and NSGA-II as multi-objective approach. For each evaluation scenario detailed in Section 4.4, the table presents a row for every feature combination under consideration. It should be reminded that in the dark cell scenario, we computed the metrics only for two (Curv-MLP and Curv-TurnCnt) feature combinations, since there was not any dark cell in the TurnCnt-StdSA AMP (see Section 4).

For dark and grey targets, NSGA-II always produced the highest number of diverse inputs in close proximity to the target (i.e., TC and TCD values).

For white targets, NSGA-II achieved the highest TC, TT, TCD and TTD values for two feature combinations out of three. Remarkably, NSGA-II was the only DeepAtash-LR configuration able to generate inputs in proximity of the white target for the TurnCnt-StdSA feature combination.

These findings highlight overall the effectiveness of both search strategies (DeepAtash-LR with GA vs NSGA-II), with an advantage observed in favor of NSGA-II. The last row of Table 5 shows that on average, NSGA-II generated a higher number of diverse misbehaviours both close to and on the target cell.

The statistical significance of this performance difference was observed in all metrics, except for the test on the target diversity (TTD) metric, for which both strategies achieved statistically comparable results. By employing the NSGA-II algorithm, our tool optimized multiple fitness functions simultaneously. Such a multi-objective approach allowed for a more comprehensive exploration of the feature space. Consequently, this led to an increased number of useful test inputs and provided a diverse set of driving scenarios.

Summary: *The multi-objective configuration of DeepAtash-LR generated a larger number of inputs in close proximity to and exactly on the target cell compared to the single-objective configuration. Moreover, the multi-objective configuration exhibited higher diversity of the generated inputs.*

Table 6. RQ3 - Results Achieved by the Compared Tools

		DeepAtash-LR		DeepHyperion-CS	
	Features	TC [TCD]	TT [TTD]	TC [TCD]	TT [TTD]
Dark	Curv-MLP	**6.70** [**0.64**]	**5.30** [**0.68**]	0.90 [0.10]	0.10 [0.02]
	Curv-TurnCnt	**7.40** [**0.79**]	**6.40** [**0.78**]	1.50 [0.40]	0.10 [0.02]
Grey	Curv-MLP	**5.80** [**0.60**]	**4.60** [**0.70**]	2.30 [0.31]	0.10 [0.10]
	Curv-TurnCnt	**4.10** [**0.55**]	**0.70** [**0.10**]	1.20 [0.2]	0.10 [0.05]
	TurnCnt-StdSA	**3.70** [**0.70**]	**1.70** [**0.50**]	0.40 [0.25]	0.10 [0.10]
White	Curv-MLP	**4.70** [0.48]	**4.70** [**0.50**]	1.00 [**0.52**]	0.00 [0.00]
	Curv-TurnCnt	**7.80** [**0.97**]	**7.20** [**0.90**]	0.30 [0.12]	0.00 [0.00]
	TurnCnt-StdSA	**2.00** [**0.50**]	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]
	AVG	**4.86** [**0.59**]	**3.62** [**0.47**]	0.95 [0.30]	0.06 [0.08]

Tests close to target (TC) and their diversity (TCD); tests on target (TT) and their diversity (TTD). In each row, boldface indicates the maximum of each of the four metrics; underline indicates values significantly higher than the remaining ones (p-value < 0.05, non-negligible effect size).

5.4 RQ3: Comparison

Table 6 reports the results achieved by DeepAtash-LR and the existing approach DeepHyperion-CS. In this comparison, we focused on the DeepAtash-LR configuration featuring the NSGA-II multi-objective search strategy, as it had demonstrated superior performance in the previous research question.

As indicated in the first two rows, for all dark targets, DeepAtash-LR demonstrated its superiority by generating significantly more and more diverse inputs in close proximity to and exactly on the target cell. Remarkably, for Curv-MLP and Curv-TurnCnt feature combinations, DeepAtash-LR generated an average of 58.5× more misbehaviours on target compared to DeepHyperion-CS.

For most of the grey targets, DeepAtash-LR and DeepHyperion-CS showed statistically comparable performance. Notably, DeepAtash-LR outperformed DeepHyperion-CS by generating a significantly higher number of tests close and on the target for TurnCnt-StdSA and tests on the target for the Curv-MLP feature combinations, along with achieving higher diversity. This performance gap was statistically significant for TT (46× higher) and TTD (7× higher).

As regards targets for which DeepHyperion-CS was unable to generate failure-inducing inputs (i.e., white targets), DeepAtash-LR produced an average of up to 7.20 such inputs. This highlights DeepAtash-LR's ability to successfully cover regions in the feature space that were completely unexplored by DeepHyperion-CS. Specifically, for the TurnCnt-StdSA feature combination, DeepHyperion-CS failed to generate any misbehaviours even in close proximity to the target, while DeepAtash-LR succeeded in producing an average of two diverse misbehaviour-inducing inputs.

The comparisons carried out across various evaluation scenarios consistently show that DeepAtash-LR outperformed DeepHyperion-CS, with statistically significant superiority observed in 60% of these comparisons. In the final row of Table 6, it is evident that, on average, DeepAtash-LR achieved better results than DeepHyperion-CS. The statistical significance of this performance difference was observed across all the considered metrics. In particular, DeepAtash-LR generated an impressive 60 times more inputs for the selected targets, underlining its substantial advantage. Our results demonstrate the capability of DeepAtash-LR in generating test inputs in feature space areas where the test generator DeepHyperion-CS can generate few or no inputs.

Table 7. RQ4 - Mean Absolute Error (MAE), Mean Squared Error (MSE) and Success Rate (SR) on the Original Test Set and on the Test Set Generated by DeepAtash-LR, Before and After Fine Tuning the DL System with the Training Partition of Generated Inputs

	Test Set		DA Test Set	
Features	MAE before	MAE after	MAE before	MAE after
Curv-MLP		**0.0186**	0.1035	**0.0797**
Curv-TurnCnt	0.0245	**0.0192**	0.1499	**0.1026**
TurnCnt-StdSA		**0.0193**	0.1219	**0.1012**
	MSE before	MSE after	MSE before	MSE after
Curv-MLP		**0.0003**	0.0150	**0.0095**
Curv-TurnCnt	0.0006	**0.0003**	0.0301	**0.0139**
TurnCnt-StdSA		**0.0003**	0.0222	**0.0159**
	SR before	SR after	SR before	SR after
Curv-MLP		0.99		**0.75**
Curv-TurnCnt	**1.00**	0.93	0.00	**0.57**
TurnCnt-StdSA		**1.00**		**0.58**

In each row, boldface indicates the minimum for MAE/MSE and maximum for SR; underline indicates values significantly lower/higher than the remaining ones (p-value < 0.05, non-negligible effect size).

Summary: *DeepAtash-LR outperforms the state-of-the-art tool DeepHyperion-CS in generating misbehaviour-inducing inputs with specific target feature value combinations. DeepAtash-LR can effectively explore crucial areas of the feature space of the DL system, which might be overlooked by DeepHyperion-CS.*

5.5 RQ4: Usefulness

Table 7 shows the improvements of the driving agent, which were achieved through fine tuning using $training_{DA}$, the training partition of the inputs generated by DeepAtash-LR. The improvement is assessed by using two different test sets: (1) the test set made of 10 random roads and (2) $test_{DA}$, the test set partition generated by DeepAtash-LR. The "before" columns display the performance of the DL system on these two test sets; the "after" columns display the performance of the fine-tuned DL system.

Since we selected a high-quality ADS, its initial MSE on the original test set was quite low. Consequently, it was able to perfectly handle the driving task on such roads, achieving a SR of 1. On the other hand, the ADS's initial MSE on $test_{DA}$ was higher and its success rate was obviously 0, as this set consisted of misbehaviour-inducing inputs.

Table 7 (top) and (middle) report the results of the offline evaluation, in terms of *MAE* and *MSE*. Across all feature combinations, the fine-tuned ADS exhibited an improvement in its prediction accuracy. In fact, both *MAE* and *MSE* significantly diminished after the fine tuning process, for both the test set and $test_{DA}$. Quite surprisingly, we improved the *MAE*/*MSE* also on the original test set despite the system's initial high quality. Therefore, we not only witnessed the absence of any signs of regression during the offline validation, but also noticed a slight improvement in MAE/MSE on the original test set. This might be due to an increased generalization capability induced by the additional training on inputs with under-represented feature combinations. Table 7 (bottom) reports the results of the evaluation in the simulation loop, in terms of SR. After fine tuning, we notice minimal regression in terms of the system's ability to successfully complete the driving task. In fact, the SR on the test set remains notably high and is statistically comparable to the perfect score achieved before the fine tuning process. On the other hand, the fine-tuned ADS demonstrated a significant improvement in its ability to drive on the roads belonging to $test_{DA}$ (i.e., SR = 63% on average).

Summary: *DeepAtash-LR is useful to improve the performance of the ADS through fine tuning, by targeting feature combinations under-represented or unseen during development. The inputs generated by our tool can serve as a diverse training set, facilitating the enhancement of the performance of the ADS without compromising its success rate.*

5.6 Threats to Validity

External Validity: The selection of the experimental subject may pose a potential threat to external validity. Nevertheless, we mitigated this concern by choosing an ADS widely employed in SE research and adopting a driving simulator with precise physics simulation capabilities. Furthermore, we verified that our chosen subject could successfully manage all the driving tasks within a randomly generated test set.

The choice of targets introduces another potential external validity threat, as the obtained results may not necessarily generalize to different target selections. To address and mitigate this threat, we adopted a strategy for selecting three distinct types of targets, each corresponding to different usage scenarios.

Internal Validity: The usage of LR and DNN may not be representative of all possible surrogate models. LRs have small number of parameters and can be trained with small sample of training data. DNNs are more general, as they can approximate any non linear function, but they have more parameters to train and, correspondingly, require more training data and time. Therefore, we believe that these two models cover the two most interesting classes of surrogate models, although we acknowledge that alternatives do exist.

Conclusion Validity: The stochastic nature of DL-based ADSs and search-based approaches may affect the results. To address this concern, we adopted a rigorous experimental methodology by running each experiment multiple times and conducting standard statistical tests to assess the significance of the obtained results.

Reproducibility: Our results can be replicated using the replication package and experimental data we have made available for DeepAtash-LR [63, 64].

6 RELATED WORK

Although focused test case generation has been a subject of extensive study and application in the context of software testing, its application to ADSs is a new field that has been explored only in our previous work, DeepAtash [62]. Hence, we organize the related works according to two main themes: focused test generation for traditional software and (unfocused) test generation specifically tailored for ADSs.

6.1 Automated Focused Test Generation

In the SE literature, numerous approaches have been proposed for the automated generation of test inputs, especially for software testing purposes. The main goal of these approaches is to efficiently exercise the functionality of the software under test, while also aiming to reveal any fault that may exist within the software [59]. Among these approaches, search-based techniques have demonstrated their effectiveness and efficiency, particularly when dealing with problems that have complex input spaces. Indeed, search-based methods are well-suited for scenarios where an exhaustive or symbolic analysis would be impractical due to the size of the input space and the complexity of the involved input constraints [37].

The empirical study conducted by Shamshiri et al. [49] highlighted a significant challenge in software testing. Despite achieving extensive code coverage, many state-of-the-art test generators still experienced a low fault detection rate. The main reason for this issue is that merely covering faulty code is often insufficient to trigger a failure, as specific inputs or scenarios are required to expose the faults. Focused test generation approaches offer a promising solution to this problem. By generating diverse test inputs that target specific parts of the software under test, these approaches can effectively increase the likelihood of triggering failures.

Alipour et al. [4] introduce *directed swarm testing*, which enhances traditional swarm testing methods by generating tests with an increased probability of covering specific source code targets. By intelligently directing the swarm testing process, this approach aims to prioritize the exploration of critical areas in the software code, improving the effectiveness of test case generation.

Gotlieb and Petit [20] propose a technique that leverages constraint solving to achieve a specific control flow path coverage for the program under test. By formulating constraints based on the desired control flow path and using a constraint solver, this approach can systematically generate test cases that traverse the specified path.

The **DFT (Diversified Focused Testing)** approach proposed by Menéndez et al. [38] presents a testing strategy that combines model checking and search-based testing techniques. The main objective of this approach is to generate test inputs that effectively reach specific program points of interest while simultaneously promoting input diversity.

The IFRIT approach, introduced by Romdhana et al. [47], is a novel testing technique that harnesses the power of **Reinforcement Learning (RL)** to generate diverse test inputs targeting specific points in the program under test. IFRIT's RL agent rewards the generated inputs if they are diverse from the previously generated solutions and they are able to cover the desired program locations.

While the aforementioned approaches were proposed for traditional software, we tailor DeepAtash to the unique characteristics and challenges of ADSs. In particular, we use a surrogate model to minimize the need for executing expensive simulations to get the fitness values required by search-based algorithms.

6.2 Test Generation for Autonomous Driving Systems

Test generation is crucial for ensuring the reliability of ADSs. These systems operate in complex and dynamic environments, making it challenging to test all possible scenarios. Several techniques and approaches have been developed to address this challenge.

DeepXplore is a test generator guided by neuron coverage, i.e., the number of activated neurons. In particular, a neuron is considered activated if its output value is higher than a predefined threshold. DeepXplore exclusively tests ADSs in an offline setting, i.e., maximizing the MSE of the steering angle prediction. Consequently, it may generate false positives for inputs that do not result in real system failures.

Other test generators for ADSs, such as AsFault [19], DeepJanus [46], DoppelTest [26], or GenBo [12], operate within an online setting, i.e., they simulate the behaviour of the vehicle in the scenarios they generate, as done by DeepAtash-LR. Unlike focused test generators, these tools do not explicitly target specific feature space areas. Moreover, they may spend considerable time in performing unnecessary simulations, since they lack the aid of a surrogate model.

NSGAII-DT [2] is a test generator designed for vision-based control systems. Similarly to DeepAtash-LR, it utilizes evolutionary algorithms. However, it targets areas within the input space that are likely to trigger system misbehaviours, while our tool, in contrast, is designed to target pre-selected feature value combinations with failure inducing and diverse inputs. Explicitly pre-selecting the targets, allows DeepAtash-LR to explore the critical feature values that lead to misbehaviours, stress the system with non-critical features, or explore the features values that are not covered yet. NSGAII-DT is guided by decision trees that are constructed based on critical combinations of structural features learned throughout the exploration process. While we also employ learning-based surrogate models trained during the search, ours are used for predicting fitness or behavioural feature values, in order to efficiently explore the input space, rather than for classifying the criticality of a feature combination as NSGAII-DT does.

AmbieGen [29] is a search-based framework for generating diverse misbehaviour-inducing test scenarios for ADSs. Like DeepAtash-LR, it explicitly promotes test diversity and employs a simplified system model to approximate results without running time-consuming simulations.

SAMOTA (Surrogate-Assisted Many-Objective Testing Approach) [23] is a testing approach that combines many-objective search and surrogate-assisted optimization techniques. Its approach consists of two search phases, global search using global surrogate models to explore the search space and capture the global fitness landscape, and local search using local surrogate models to exploit promising areas found by the global search. Unlike our approach, each objective of SAMOTA is a safety violation, and the specific features of the misbehaviour-inducing inputs are not relevant for SAMOTA. More specifically, the goal of SAMOTA is to trigger a safety violation with any combination of driving scenario features that the surrogate model predicts as highly likely to produce a misbehaviour. On the other hand, DeepAtash-LR's main objective is to generate driving scenarios that are focused on a specific feature combination of interest, while triggering a misbehaviour is a secondary objective of the search process. While both search processes benefit from a surrogate model, which makes them more efficient, the role of the surrogate model in the two search algorithms is different, as in one case (SAMOTA) it aims primarily at exposing misbehaviours, while in the other (DeepAtash-LR) it aims primarily at covering a specific (e.g., previously non-visited) feature map cell. The authors of SAMOTA [23] show the usefulness of a surrogate model for misbehaviour exposure; we show its usefulness for focused feature map coverage (the latter does not necessarily descend from the former).

Among the testing approaches for ADSs mentioned earlier, some of them adopt surrogate models to increase test generation efficiency and all of them take into account various aspects such as the environment and dynamic elements. However, none of them specifically focuses the search for test scenarios on interesting, human-interpretable combinations of structural and behavioural features. In contrast, DeepAtash-LR stands out as the first surrogate-assisted tool that generates test inputs focused on a specific combination of human-interpretable features.

7 CONCLUSIONS AND FUTURE WORK

In this work, we proposed DeepAtash-LR, a novel focused test generator for ADSs. DeepAtash-LR employs a surrogate model as a proxy for an actual system's execution, thus sidestepping the need for resource-intensive evaluations, which would involve complete simulations of the driving tasks on the candidate test scenarios. Our empirical study shows that the evaluation using surrogate model implemented within DeepAtash-LR significantly improves the effectiveness of DeepAtash in generating misbehaviour-inducing inputs in the proximity of the target features. Moreover, we obtained empirical evidence showing that the focused inputs generated by DeepAtash-LR can be useful for improving the ADS through fine tuning. Despite the positive results achieved by integrating the Linear Regression surrogate model into our focused test generation, we plan to explore alternative surrogate models and algorithms in the future. For example, we may consider retraining the surrogate model after regular intervals with new data.

In our future work, we also plan to generalize our results to additional ADSs, including industrial ones. Moreover, we plan to extend our approach to other cyber-physical systems, such as umanned aerial vehicles, e.g., drones.

ACKNOWLEDGMENTS

The driving simulator has been kindly provided by BeamNG GmbH.

REFERENCES

[1] 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 *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*. ACM, 63–74. https://doi.org/10.1145/2970276.2970311

[2] 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, Gothenburg, Sweden, 1016–1026. https://doi.org/10.1145/3180155.3180160

[3] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing autonomous cars for feature interaction failures using many-objective search. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*. ACM, Montpellier, France, 143–154. https://doi.org/10.1145/3238147.3238192

[4] Mohammad Amin Alipour, Alex Groce, Rahul Gopinath, and Arpit Christi. 2016. Generating focused random tests using directed swarm testing. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*. 70–81.

[5] Andrea Arcuri and Lionel Briand. 2014. A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability* 24, 3 (2014), 219–250. https://doi.org/10.1002/stvr.1486

[6] David Arthur and Sergei Vassilvitskii. 2006. *k-means++: The Advantages of Careful Seeding*. Technical Report. Stanford.

[7] Phillip J. Barry and Ronald N. Goldman. 1988. A recursive evaluation algorithm for a class of Catmull-Rom splines. *SIGGRAPH Comput. Graph.* 22, 4 (June 1988), 199–204. https://doi.org/10.1145/378456.378511

[8] BeamNG GmbH. 2018. *BeamNG.research*. BeamNG GmbH. https://www.beamng.gmbh/research

[9] Yoshua Bengio. 2011. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27 (UTLW'11)*. JMLR.org, Washington, USA, 17–37.

[10] Matteo Biagiola, Stefan Klikovits, Jarkko Peltomaki, and Vincenzo Riccio. 2023. SBFT tool competition 2023 - cyber-physical systems track. In *16th IEEE/ACM International Workshop on Search-Based And Fuzz Testing, SBFT 2023, Melbourne, Australia, May 14, 2023.*

[11] Matteo Biagiola, Andrea Stocco, Vincenzo Riccio, and Paolo Tonella. 2023. Two is better than one: Digital siblings to improve autonomous driving testing. *arXiv preprint arXiv:2305.08060* (2023).

[12] Matteo Biagiola and Paolo Tonella. 2023. Boundary state generation for testing and improvement of autonomous driving systems. *arXiv preprint arXiv:2307.10590* (2023).

[13] Matteo Biagiola and Paolo Tonella. 2023. Testing of deep reinforcement learning agents with surrogate models. *arXiv preprint arXiv:2305.12751* (2023).

[14] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. 2016. End to end learning for self-driving cars. *CoRR* abs/1604.07316 (2016), 1–9. arXiv:1604.07316 http://arxiv.org/abs/1604.07316

[15] Edwin Catmull and Raphael Rom. 1974. A class of local interpolating splines. In *Computer Aided Geometric Design*, R. E. Barnhill and R. F. Riesenfeld (Eds.). Academic Press, 317 – 326. https://doi.org/10.1016/B978-0-12-079050-0.50020-5

[16] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197.

[17] Gordon Fraser and Andrea Arcuri. 2011. Evolutionary generation of whole test suites. In *2011 11th International Conference on Quality Software*. IEEE, 31–40.

[18] Alessio Gambi, Gunel Jahangirova, Vincenzo Riccio, and Fiorella Zampetti. [n.d.]. SBST tool competition 2022. In *15th IEEE/ACM International Workshop on Search-Based Software Testing, SBST 2022, Pittsburgh, PA, USA, May 9, 2022.*

[19] Alessio Gambi, Marc Müller, and Gordon Fraser. 2019. Automatically testing self-driving cars with search-based procedural content generation. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, Beijing, China, July 15-19, 2019*. ACM, 318–328. https://doi.org/10.1145/3293882.3330566

[20] Arnaud Gotlieb and Matthieu Petit. 2010. A uniform random test data generator for path testing. *Journal of Systems and Software* 83, 12 (2010), 2618–2626.

[21] Qinghua Gu, Qian Wang, Neal N. Xiong, Song Jiang, and Lu Chen. 2021. Surrogate-assisted evolutionary algorithm for expensive constrained multi-objective discrete optimization problems. *Complex & Intelligent Systems* (2021), 1–20.

[22] Antonio Guerriero, Roberto Pietrantuono, and Stefano Russo. 2021. Operation is the hardest teacher: Estimating DNN accuracy looking for mispredictions. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 348–358.

[23] Fitash Ul Haq, Donghwan Shin, and Lionel Briand. 2022. Efficient online testing for DNN-enabled systems using surrogate-assisted and many-objective optimization. In *Proceedings of the 44th International Conference on Software Engineering*. 811–822.

[24] 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 Software Engineering* 26, 5 (2021), 90.

[25] Geoffrey E. Hinton and Sam Roweis. 2002. Stochastic neighbor embedding. *Advances in Neural Information Processing Systems* 15 (2002).

[26] Yuqi Huai, Yuntianyi Chen, Sumaya Almanee, Tuan Ngo, Xiang Liao, Ziwen Wan, Qi Alfred Chen, and Joshua Garcia. 2023. Doppelgänger test generation for revealing bugs in autonomous driving software. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2591–2603.

[27] Nargiz Humbatova, Gunel Jahangirova, Gabriele Bavota, Vincenzo Riccio, Andrea Stocco, and Paolo Tonella. 2020. Taxonomy of real faults in deep learning systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE'20)*. Association for Computing Machinery, Seoul, South Korea, 1110–1121. https://doi.org/10.1145/3377811.3380395

[28] Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. 2021. DeepCrime: Mutation testing of deep learning systems based on real faults. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*.

[29] Dmytro Humeniuk, Foutse Khomh, and Giuliano Antoniol. 2022. A search-based framework for automatic generation of testing environments for cyber–physical systems. *Information and Software Technology* 149 (2022), 106936.

[30] Gunel Jahangirova, Andrea Stocco, and Paolo Tonella. 2021. Quality metrics and oracles for autonomous vehicles testing. In *Proceedings of 14th IEEE International Conference on Software Testing, Verification and Validation (ICST'21)*. IEEE, 194–204.

[31] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding deep learning system testing using surprise adequacy. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*. IEEE / ACM, 1039–1049. https://doi.org/10.1109/ICSE.2019.00108

[32] Zelun Kong, Junfeng Guo, Ang Li, and Cong Liu. 2020. PhysGAN: Generating physical-world-resilient adversarial examples for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14254–14263.

[33] Kiran Lakhotia, Mark Harman, and Phil McMinn. 2007. A multi-objective approach to search-based test data generation. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO'07)*. ACM, London, England, 1098–1105. https://doi.org/10.1145/1276958.1277175

[34] Craig Larman. 1997. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*. Prentice Hall.

[35] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

[36] Ke Mao, Mark Harman, and Yue Jia. 2016. Sapienz: Multi-objective automated testing for Android applications. In *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016)*. ACM, Saarbrücken, Germany, 94–105. https://doi.org/10.1145/2931037.2931054

[37] Phil McMinn. 2004. Search-based software test data generation: A survey. *Software Testing, Verification and Reliability* 14, 2 (2004), 105–156.

[38] Héctor D. Menéndez, Gunel Jahangirova, Federica Sarro, Paolo Tonella, and David Clark. 2021. Diversifying focused testing for unit testing. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 4 (2021), 1–24.

[39] Jean-Baptiste Mouret and Jeff Clune. 2015. Illuminating Search Spaces by Mapping Elites. arXiv:1504.04909 [cs.AI]

[40] Shiva Nejati, Lev Sorokin, Damir Safin, Federico Formica, Mohammad Mahdi Mahboob, and Claudio Menghi. 2023. Reflections on surrogate-assisted search-based testing: A taxonomy and two replication studies based on industrial ADAS and Simulink models. *Information and Software Technology* (2023), 107286.

[41] Vuong Nguyen, Stefan Huber, and Alessio Gambi. 2021. SALVO: Automated generation of diversified tests for self-driving cars from existing maps. In *2021 IEEE International Conference on Artificial Intelligence Testing (AITest)*. IEEE, 128–135.

[42] 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.

[43] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*. Association for Computing Machinery, New York, NY, USA, 1135–1144. https://doi.org/10.1145/2939672.2939778

[44] Vincenzo Riccio, Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. 2021. DeepMetis: Augmenting a deep learning test set to increase its mutation score. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 355–367.

[45] Vincenzo Riccio, Gunel Jahangirova, Andrea Stocco, Nargiz Humbatova, Michael Weiss, and Paolo Tonella. 2020. Testing machine learning based systems: A systematic mapping. *Empir. Softw. Eng.* 25, 6 (2020), 5193–5254. https://doi.org/10.1007/s10664-020-09881-0

[46] Vincenzo Riccio and Paolo Tonella. 2020. Model-based exploration of the frontier of behaviours for deep learning system testing. In *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'20)*. Association for Computing Machinery, 13 pages. https://doi.org/10.1145/3368089.3409730

[47] Andrea Romdhana, Mariano Ceccato, Alessio Merlo, and Paolo Tonella. 2022. IFRIT: Focused testing through deep reinforcement learning. In *IEEE International Conference on Software Testing, Verification and Validation (ICST'22)*. IEEE.

[48] Peter J. Rousseeuw. 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics* 20 (1987), 53–65.

[49] Sina Shamshiri, René Just, José Miguel Rojas, Gordon Fraser, Phil McMinn, and Andrea Arcuri. 2015. Do automatically generated unit tests find real faults? An empirical study of effectiveness and challenges (T). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 201–211. https://doi.org/10.1109/ASE.2015.86

[50] Andrea Stocco, Paulo J. Nunes, Marcelo D.'Amorim, and Paolo Tonella. 2022. ThirdEye: Attention maps for safe autonomous driving systems. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.

[51] Andrea Stocco, Brian Pulfer, and Paolo Tonella. 2022. Mind the gap! A study on the transferability of virtual vs physical-world testing of autonomous driving systems. *IEEE Transactions on Software Engineering* (2022).

[52] Andrea Stocco, Brian Pulfer, and Paolo Tonella. 2023. Model vs system level testing of autonomous driving systems: A replication and extension study. *Empirical Software Engineering* 28, 3 (2023), 73.

[53] Andrea Stocco and Paolo Tonella. 2020. Towards anomaly detectors that learn continuously. In *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. 201–208. https://doi.org/10.1109/ISSREW51248.2020.00073

[54] Andrea Stocco, Michael Weiss, Marco Calzana, and Paolo Tonella. 2020. Misbehaviour prediction for autonomous driving systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 359–371.

[55] Shuncheng Tang, Zhenya Zhang, Yi Zhang, Jixiang Zhou, Yan Guo, Shuang Liu, Shengjian Guo, Yan-Fu Li, Lei Ma, Yinxing Xue, and Yang Liu. 2023. A survey on automated driving system testing: Landscapes and trends. *ACM Trans. Softw. Eng. Methodol.* 32, 5, Article 124 (Jul. 2023), 62 pages. https://doi.org/10.1145/3579642

[56] Chakkrit Tantithamthavorn and Jirayus Jiarpakdee. 2021. Monash University. https://doi.org/10.5281/zenodo.4769127 Retrieved 2021-05-17.

[57] Mark Utting, Alexander Pretschner, and Bruno Legeard. 2012. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability* 22, 5 (2012), 297–312.

[58] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, 11 (2008).

[59] Qian Yang, J. Jenny Li, and David M. Weiss. 2009. A survey of coverage-based testing tools. *Comput. J.* 52, 5 (2009), 589–597.

[60] 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*. Virtual, Denmark, 79–90.

[61] Tahereh Zohdinasab, Vincenzo Riccio, Alessio Gambi, and Paolo Tonella. 2022. Efficient and effective feature space exploration for testing deep learning systems. *ACM Trans. Softw. Eng. Methodol.* (Jun. 2022). https://doi.org/10.1145/3544792 Just Accepted.

[62] Tahereh Zohdinasab, Vincenzo Riccio, and Paolo Tonella. 2023. DeepAtash: Focused test generation for deep learning systems. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. Association for Computing Machinery, 954–966.

[63] Tahereh Zohdinasab, Vincenzo Riccio, and Paolo Tonella. 2023. DeepAtash-LR:Replication Package. https://github.com/testingautomated-usi/DeepAtash

[64] Tahereh Zohdinasab, Vincenzo Riccio, and Paolo Tonella. 2023. DeepAtash-LR:Replication Package. https://doi.org/10.5281/zenodo.10441955

[65] Tahereh Zohdinasab, Vincenzo Riccio, and Paolo Tonella. 2023. An empirical study on low- and high-level explanations of deep learning misbehaviours. In *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'23)*. 1–11. https://doi.org/10.1109/ESEM56168.2023.10304866

Received 10 September 2023; revised 2 April 2024; accepted 10 April 2024