The Journal of Systems and Software 212 (2024) 112017

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

# Semantic-guided fuzzing for virtual testing of autonomous driving systems ☆

An Guo [a], Yang Feng [a,*], Yizhen Cheng [a], Zhenyu Chen [a,b]

[a] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210093, China
[b] Shenzhen Research Institute of Nanjing University, Shenzhen, China

ARTICLE INFO

*Keywords:*
Software testing
Fuzz testing
Driving scenario description language
Autonomous driving system

ABSTRACT

Autonomous driving systems (ADS) have achieved spectacular development and have been utilized in numerous safety-critical tasks. Nonetheless, in spite of their considerable advancement, ADS perception components with high complexity and low interpretability often demonstrate unexpected corner-case behaviors. Several real-world accidents involving self-driving cars even lead to fatalities. Before rolling the autonomous vehicles out to the end-users, it is vital to test and guarantee the safety of ADS. As one of the most critical autonomous driving testing techniques, the prevailing virtual testing depends on the tester using tool-specific languages to code traffic simulation programs correctly. However, this process often requires plenty of effort, and it may fail to capture various rare events from complex driving situations that require sophisticated awareness of the surroundings.

In this paper, we design and implement a semantic-guided scene fuzzing framework for autonomous driving systems, namely FuzzScene, based on the metamorphic testing theory. It employs driving scenario description language for scenario representation and equips a tree-based mutation strategy to generate tests with proper oracle information. To improve the testing efficiency and detect misbehaviors under different settings, we propose a unique sampling strategy and construct a testing guidance criterion to optimize FuzzScene. We experiment FuzzScene with multiple steering controllers to evaluate its performance on different tasks. The experiment results show that the semantic transformed driving scenarios generated by FuzzScene efficiently detect hundreds of inconsistent behaviors of ADS. Also, the results confirm that FuzzScene can improve steering precision by retraining with the generated scenes.

## 1. Introduction

The rapid development of artificial intelligence and sensing technology over the past decade has made autonomous vehicles around the corner. As a typical intelligent software, autonomous driving systems (ADS) have been designed and deployed to assist in many safety-critical tasks, such as transit and delivery service (Japan, 2022), self-driving taxi (Baidu, 2022), and other fields. However, such intelligent systems often exhibit incorrect and unexpected corner-case behaviors, which can cause fatal collisions when deployed in real-world environments. The National Highway Traffic Safety Administration (NHTSA) received reports of 130 crashes involving vehicles equipped with ADS from July 2021 to May 2022 (US, 2022). Therefore, the safety of autonomous driving systems raises wide concerns, and it is imperative to conduct comprehensive testing to ensure their safety and reliability.

The primary quality assurance strategies of ADS can be categorized into two families: real-world testing and virtual testing. Real-world testing entails testing specific driving scenarios in closed autonomous vehicle proving grounds (Lou et al., 2022) or monitoring autonomous vehicles in real traffic (Zhao et al., 2019), but this approach requires a long period and extensive resources. For instance, a previous study concluded that it needs a fleet of 100 autonomous vehicles driving continuously for around 500 years to statistically prove their reliability (Kalra and Paddock, 2016).

Moreover, public road testing may fail to test against realistic variations of corner cases. On the other hand, in recent years, high-fidelity virtual testing has become essential to the development and validation of autonomous driving technologies (Li et al., 2019; Abeysirigoonawardena et al., 2019). It conducts autonomous vehicle testing in simulation platforms, such as CARLA (Dosovitskiy et al., 2017) and LGSVL (Rong et al., 2020). Instead of simply calculating total driving mileage in a simulated world, we usually expose the system under test to defined driving scenarios (Junietz et al., 2018; Sun et al.,

☆ Editor: Antonia Bertolino.

* Corresponding author.
*E-mail addresses:* guoan218@smail.nju.edu.cn (A. Guo), fengyang@nju.edu.cnn (Y. Feng), yz_cheng@smail.nju.edu.cn (Y. Cheng), zychen@nju.edu.cn (Z. Chen).

https://doi.org/10.1016/j.jss.2024.112017
Received 17 August 2023; Received in revised form 8 February 2024; Accepted 4 March 2024
Available online 16 March 2024

2022). By customizing these test scenarios with varying distributions, ADS test engineers can more effectively evaluate rare and extreme scenarios and avoid wasting time on repeated scenarios. However, the ego car interacts with its diverse operating environment, including the driving path, weather conditions, and other moving agents. Simulating potential virtual crash scenarios in such an environment is a nontrivial task. Typically, key parameters that can describe a driving scenario are selected, and a parameter space is established. From this parameter space, specific scenarios with defined parameter values are extracted for testing and evaluation. Nonetheless, due to a large number of possible scenario parameters, the combinations of parameters explode when generating concrete scenarios in simulations (Gao et al., 2019). The primary challenge is how to optimize testing efficiency. To speed up the virtual testing process, accelerated testing methods that aim to identify safety-critical scenarios are necessary.

For both conventional and DNN-based software, fuzz testing (Miller et al., 2022; Li et al., 2020; Braiek and Khomh, 2020) is a widely used approach to identify bugs by navigating extensive search spaces. At a high level, fuzzing mutates existing inputs to create tests with the goal of exploring new bugs. However, integrating fuzzing into the virtual testing of ADS is not straightforward. Firstly, driving scenarios exhibit diverse characteristics and interdependencies, and random mutations of arbitrary features may result in semantically incorrect scenarios. Furthermore, ADS test engineers require additional effort to learn tool-specific languages and simulated traffic programs from scratch. Finally, these techniques often neglect corner cases or repeatedly uncover erroneous behaviors similar to those already discovered. Thus they cannot diversify the detected incorrect behaviors.

To fill the gap, we propose a systematic semantic-guided fuzzing testing approach, called FuzzScene, for automatically testing autonomous driving systems and further retraining to improve them. FuzzScene can leverage the driving scenario description language OpenSCENARIO (ASAM, 2020) for scenario representation. Then it parses semantic information of the driving environment and generates testing scenarios with the grammar-aware generation strategy in the CARLA simulator. Moreover, it constructs a testing guidance criterion to optimize FuzzScene and utilizes transformation-specific metamorphic relations to automatically detect diverse faults by comparing transformed driving scenes to the original data.

We conducted experiments with FuzzScene to evaluate its performance using four steering controller models. We demonstrate the efficacy of our parsed semantic transformations, as applying variable semantic transformations resulted in a decrease in Mean Squared Error (MSE). Next, our proposed learning-to-rank sampling strategy and testing criteria enable the fuzzing algorithm to discover a greater number of inconsistent behaviors than the baseline methods, and these behaviors exhibit diversity. Further, the experimental results reveal that the generated scenes can significantly increase the MSE for controller models after retraining. These results indicate that FuzzScene can effectively detect diverse inconsistent behaviors and improve the robustness of autonomous driving systems through retraining with the transformed data.

The contributions of this paper can be summarized as follows:

- **Method.** We present a semantic-guided fuzzing technique, which leverages the driving scenario description language to generate realistic and valid scenarios. Specifically, we propose a novel grammar-aware generation strategy and design a testing guidance criterion that induces both failure and diversity to optimize FuzzScene in discovering a greater variety of inconsistent behaviors.
- **Tool.** We implement the proposed methods into an automated fuzz testing tool called FuzzScene, which enables ADS test engineers to efficiently discover various inconsistent behaviors. We have made the source code of FuzzScene available[1] and released the generated transformed driving scenes.
- **Study.** We conduct extensive studies to investigate the effectiveness of FuzzScene. We employ our tool to test four typical steering controller models and find various inconsistent decision results, many of which could lead to potentially fatal collisions.

## 2. Background & Motivation

### 2.1. Virtual testing of ADS

Testing ADS on physical vehicles is neither cost-effective nor safe, and it is often challenging to reproduce various types of traffic accidents. A realistic and effective alternative is to test ADS under a simulated environment (Chao et al., 2020), where driving scenes and physical logic (e.g., vehicle dynamics, weather conditions, etc.) are created in a virtual world. Virtual testing allows for checking the behavior of self-driving cars in a large number of scenarios, driver characteristics, and system configurations.

To simulate traffic situations in a testing environment, various car manufacturers and organizations developed driving simulation platforms (Kaur et al., 2021). A virtual driving simulation platform needs to provide the following main components (Schöner, 2018), including road model, traffic model, sensor model, and vehicle model. With model-based simulation, autonomous vehicles can obtain a perfect picture of the surrounding world in the driving simulator. Each simulation platform has its own characteristics; PreScan[2] has a strong capability of constructing realistic environments and simulating different weather conditions. LGSVL and CARLA are suited for end-to-end testing of unique functionalities (e.g., perception, localization, etc.). Scenario description languages can assist virtual testing of autonomous driving. The scenario description files that depict the dynamic content of driving and traffic simulators are parsed and interact with the simulator to produce corresponding scenarios. ADS test engineers are only expected to create these scenario description files, as they do not need to learn the specific details of the simulator's API grammar.

Simulation of traffic scenarios is crucial to the entire verification and validation process. However, with the increased number of driving scenario key parameters, the permutations of their values explode exponentially. One core challenge is how to explore safety-critical scenarios within huge parameter space (Wang et al., 2022). In the current literature, accelerated evaluation has been proposed to improve ADS testing efficiency. Zhang et al. (2018b) apply the importance sampling technique to enhance the exposure of rare events, which are considered safety-critical scenarios. Other types of testing methods identify safety-critical scenarios via step-by-step searching. The search direction is towards safety reduction and could be optimized using genetic algorithms (Ben Abdessalem et al., 2016) and reinforcement learning methods (Ding et al., 2021). This type of method is compatible with black-box autonomous driving control algorithms, while it holds the issue of converging to a local optimum. By designing and choosing the most appropriate test automation methods, we can significantly optimize the efficiency and effectiveness of the entire virtual testing process.

### 2.2. The driving scenario description language

In the operational environment of autonomous driving, the ADS receives multi-dimensional inputs from various sensors (e.g., cameras, LiDAR, radar, etc.) and processes semantic information to drive the car. Consequently, the driving scenarios encountered by the ADS have become a meaningful metric, leading to a shift towards a scenario-oriented testing approach in the automotive industry and research community. The creation of scenarios is usually based on various

[1] https://github.com/meng2180/FuzzScene.

[2] https://www.plm.automation.siemens.com/global/en/products/simulation-test/active-safety-system-simulation.html.

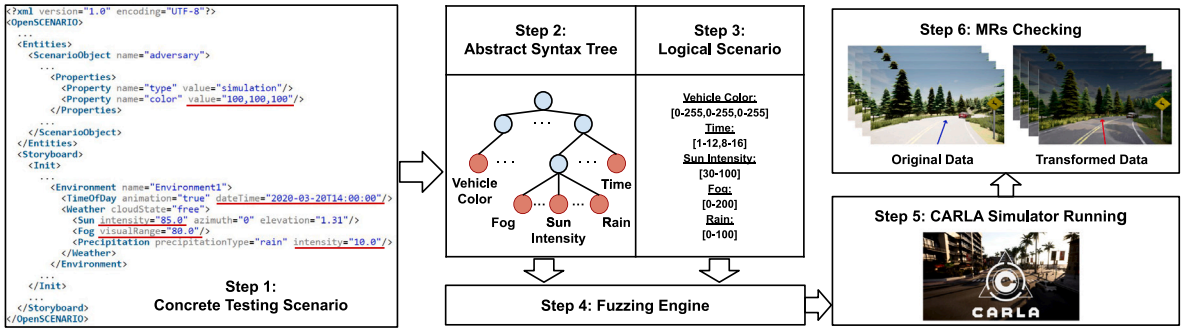


**Fig. 1.** The motivating running example of FuzzScene.

data sources (de Gelder and Paardekooper, 2017) or expert knowledge (Bagschik et al., 2018), and the execution of driving scenarios can be realized in X-in-the-loop (XiL) simulations (Gangopadhyay et al., 2019). However, it requires a standard, executable, and understandable language to describe and transfer the scenarios between the creation and execution stages. To tackle this problem, both the industry and academia have been working on designing driving scenario description languages to model complex environments for ADS (Zhang et al., 2020).

For describing the road layout and road features, popular open languages formats such as OpenStreetMap (Haklay and Weber, 2008), OpenDRIVE (Diaz-Diaz et al., 2022), and Lanelets (Bender et al., 2014) are commonly used to form the static scenery element of driving scenarios. The ASAM OpenDRIVE format provides common road network descriptions that can be exchanged between different simulators. Lanelets is more lightweight than OpenDRIVE, yet still offers a specification for highly detailed maps. Althoff et al. (2018) developed an automatic conversion tool from OpenDRIVE to the Lanelets format.

Several formal dynamic scenario description languages have been proposed for describing the dynamic contents of driving simulations at a logical level. Scenic (Fremont et al., 2019) is a probabilistic programming language for scenario specification and data generation; its domain-specific design permits concise representations of complex driving scenarios and enables specialized sampling techniques. The Measurable Scenario Description Language (M-SDL) (Measurable, 2020), introduced after the first version of Scenic, does provide declarative constraints and compositional features similar to Scenic. In contrast, GeoScenario (Queiroz et al., 2019), Paracosm (Majumdar et al., 2019) and OpenSCENARIO are not probabilistic programming languages, meaning they lack probability distributions and a declarative way of specifying geometric constraints. However, the semantic information of driving scenarios generated using non-probabilistic programming languages is more straightforward to parse and transform. OpenSCENARIO is used by various companies and organizations to create simulations and tests for autonomous vehicles (OpenSCENARIO, 2020). In this paper, we use OpenSCENARIO for driving scenario representation and the ScenarioRunner[3] module to execute traffic scenarios for the CARLA simulator.

## 3. The fuzzing framework

In this section, we present the design and implementation of FuzzScene. We describe the design ideas and steps of the entire workflow in Section 3.2 and introduce the detailed designs in Sections 3.3–3.5.

[3] https://github.com/carla-simulator/scenario_runner.

### 3.1. Definitions and motivating example

#### 3.1.1. Definitions

To give a better understanding of autonomous driving virtual testing, we introduce a few terms based on previous studies (Ulbrich et al., 2015; PEGASUS, 2019; Zhong et al., 2022b):

***Scene.*** A scene is a snapshot in the simulation that contains the detailed properties (e.g., location, velocity, steering, etc.) of the ego car, other dynamic objects, the surrounding stationary objects, weather conditions, and road conditions. For instance, the ego car at a specific map location with a speed of 10 m/s facing south on a foggy afternoon.

***Scenario.*** A scenario describes "the temporal development between several scenes in a sequence of scenes" (Ulbrich et al., 2015). Unlike a scene, a scenario spans a certain time. Two scenes may specify the same initial locations for the ego car and other moving vehicles but distinct weather conditions can result in different driving scenarios.

***Logical Scenario.*** A logical scenario describes the parameter spaces that bound the search during the fuzzing. For example, the rain semantic transformation has its parameter range limited between 0 and 100 in this paper.

***Concrete Scenario.*** A concrete scenario is a specific instance in the logical search space. It specifies a concrete value for each parameter of a logical scenario. Thus ADS test engineers can run specific scenarios in the simulator to reproduce test cases.

#### 3.1.2. Motivating example

To aid comprehension of the concepts and testing methods, we provide a running example. As shown in Fig. 1, test engineers use OpenSCENARIO to write a concrete testing scenario in XML format. After parsing into an abstract syntax tree (AST), they define metamorphic relations (MRs) by analyzing semantic transformations and form a logical scenario describing parameter space. Next, the fuzz engine utilizes the AST and logical scenario as input, feeding the output into the CARLA simulator for execution. The generated transformed data are input into an end-to-end controller model with perception and planning capabilities, and then the model outputs steering angle decision information. Finally, the original and transformed outputs are then checked by FuzzScene to determine whether the metamorphic relation is satisfied.

### 3.2. Overview of FuzzScene

In traditional fuzz testing, the fuzzer usually maintains a seed input corpus and repeatedly mutates these inputs to generate new test data. Then, after the test cases are executed, it decides whether to keep the mutated inputs based on test criteria such as code coverage. Eventually, the fuzzer terminates continuous iteration due to reaching a particular goal or a timeout.

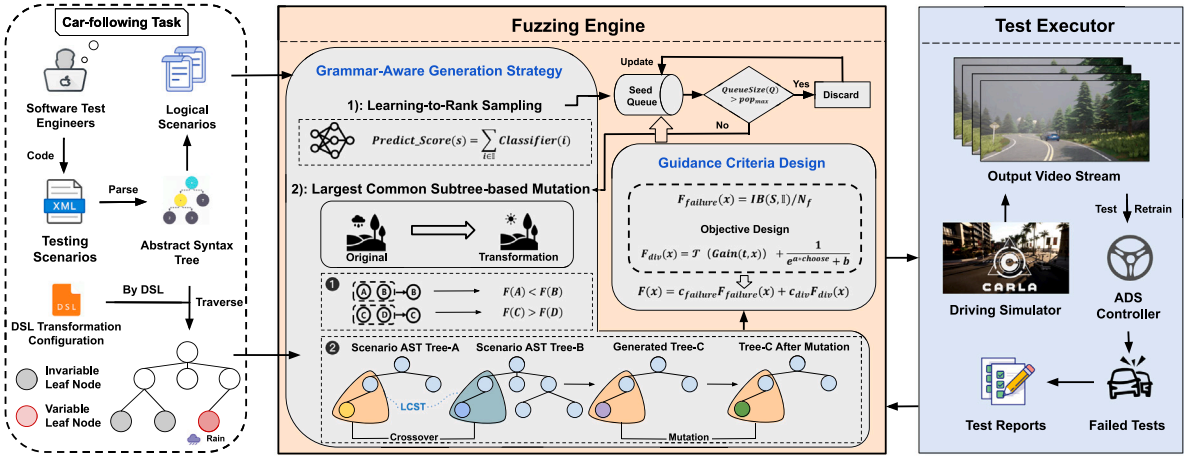


**Fig. 2.** The workflow of FuzzScene.

**Table 1**
Variable leaf nodes used by FuzzScene for generating new driving scenarios.

| Semantic transformations | Parameters | Parameter ranges |
|---|---|---|
| Vehicle color | $(c_r,\ c_g,\ c_b)$ | $c_r \in [0, 255]$, $c_g \in [0, 255]$, $c_b \in [0, 255]$ |
| Time | $(t_x,\ t_y)$ | $t_x \in [1, 12]$, $t_y \in [8, 16]$ |
| Sun intensity | $s_i$ | $s_i \in [30, 100]$ |
| Fog | $v_f$ | $v_f \in [0, 200]$ |
| Rain | $r_n$ | $r_n \in [0, 100]$ |

The fuzzing technique can be applied in the autonomous driving domain. Considering the semantic transformation parameter space is prohibitively large, we propose a fuzzing framework for ADS to detect diverse inconsistent behaviors. In order to parse the environment semantics and generate syntax-valid test scenarios, we convert the driving scenario to ASTs. Moreover, we design a testing guidance criterion to optimize FuzzScene with the aim of finding more diverse behaviors. As depicted in Fig. 2, the implementation of FuzzScene contains two major components: fuzzing engine and test executor.

ADS test engineers use scenario description language OpenSCENARIO to write testing scenarios for a specific driving task. Then testing scenarios are parsed into an AST based on the language grammar. We design logical scenarios from the AST structure and leverage them as inputs to the fuzzing engine. Meanwhile, we traverse the abstract syntax tree and obtain variable leaf nodes representing semantics according to the domain-specific language (DSL) transformation configuration file. Next, the fuzz engine efficiently leverages the grammar-aware generation strategy and guidance criteria to find combinations of semantic transformations that result in more failures. After that, the transformed video frames can be generated from CARLA to evaluate the robustness of the steering controller of autonomous driving. Finally, we achieve test reports and give feedback on semantic transformation configuration. Additionally, FuzzScene can enhance the controller's performance by retraining with the transformed data.

Specifically, the core process of the fuzzing engine is shown in Algorithm 1. For given logical scenarios $\mathbb{L}$ and DSL transformation configuration $\mathbb{T}$, users can set predefined hyperparameters $\mathbb{C}$, generations $Gens$ and the maximum queue size $pop_{max}$. The fuzzing engine first adopts a learning-to-rank strategy to sample seeds from the entire input space and put them into the seed priority queue $Q$ (Line 3). In each iteration, it selects a batch according to a seed selection method (Line 9). To generate realistic and valid concrete scenarios, the fuzzing engine generates new transformed ASTs after applying the largest common subtree-based mutation operations to the selected test cases (Line 10–19); the information on variable leaf nodes is detailed in Table 1, which contains driving semantic information such as rain and vehicle color.

**Algorithm 1:** Semantic-guided Scene Fuzzing

**Input :** the tested model $S$, DSL transformation configuration $\mathbb{T}$, designed logical scenarios $\mathbb{L}$, predefined hyperparameters $\mathbb{C}$, maximum queue size $pop_{max}$, generations $Gens$
**Output:** diverse inconsistent behaviors $\mathbb{D}$

```
1  Q ← ∅;
2  InitializeFuzzer(𝕋, ℂ, S, pop_max, Gens);
3  seeds ← LearningtoRankSampling(𝕃,𝕋);
4  Q.append(seeds);
5  foreach seed ∈ Q do
6  |  InitSeed(seed);
7  end
8  for iterate ← 1 to Gens do
9  |  selected_seeds ←TournamentSelection(Q);
10 |  i ← 1;
11 |  while i ≤ len(selected_seeds) do
12 |  |  pop_a, pop_b ←LCSTMuation(t_i, t_{i+1});
13 |  |  Q.append(pop_a);
14 |  |  Q.append(pop_b);
15 |  |  i ← i + 2;
16 |  end
17 |  foreach seed ∈ Q do
18 |  |  UpdateObj(seed);
19 |  end
20 |  if QueueSize(Q) > pop_max then
21 |  |  UpdateQueue(Q);
   |  |  // maintain queue size
22 |  end
23 end
24 return 𝔻
```

Then it runs simulations in CARLA with the supplied scenarios. Note that the failures found are recorded, and the new tests are added to the seed queue. Finally, it updates the objective values of all seeds in the queue based on the proposed objective functions (Line 21). During the whole fuzzing process, it maintains the priority queue based on the updated objective values of the seeds. The corresponding seeds with lower values are discarded if the current population is greater than $pop_{max}$ (Line 24).

### 3.3. Grammar-aware generation strategy

Self-driving cars rely on semantic information about the surrounding environment to form a complete picture of the driving situation. However, it can be challenging for the ego vehicle to understand scenes correctly and make the right decision in real-time. Even slight variations in traffic situations, such as changing an object's color, can potentially result in various traffic violations being reported. To be explicitly aware of the grammar and thus produce syntax-valid test inputs, we utilize the grammar knowledge and design a grammar-aware

generation strategy, which works at the level of ASTs. AST actually models a test input as elements with named attributes and is designed to represent all the information about a test input. Thus, ASTs provide a suitable granularity for a fuzzer to generate test inputs.

#### 3.3.1. Learning-to-rank sampling

In order to automatically find suitable initial attribute values of ASTs when sampling seed test cases, we propose a learning-to-rank sampling strategy for fuzz testing to uncover more inconsistent behaviors.

Our approach constructs a training set capturing data from CARLA for learning-to-rank, where each instance is an input of the steering model under evaluation. For each instance, we label it as 0 or 1 depending on whether the input is incorrectly predicted by the model under test. Then we train a binary classifier for each logical scenario to minimize the cross-entropy loss. Given a sampling seed $s$, the corresponding classifier predicts whether each intercepted video frame will fail or not. For the image stream $\mathbb{I}$ captured from scenario $s$, we accumulate the model outputs as the prediction score, which is defined as:

$$Predict\_Score(s) = \sum_{i \in \mathbb{I}} Classifier(i) \tag{1}$$

In the learning-to-rank sampling process, we randomly instantiate $gen_m$ candidate seed test cases for each logical scenario. Subsequently, our learning-to-rank strategy prioritizes $gen_m$ candidate seeds according to their descending scores and selects the top $q_r$ seeds. Finally, we save all selected seeds into the seed queue.

#### 3.3.2. Largest common subtree-based mutation

As the number of iterations increases, it is necessary to apply some seed data selection and mutation strategies to sample seed data for transformations. In this paper, we design a tree-based mutation algorithm to be explicitly aware of the grammar to generate syntax-valid scenarios. Specifically, our proposed largest common subtree-based (LCST-based) mutation strategy facilitates the incorporation of similar semantic information from two driving scenarios.

First, we use the binary tournament selection method (Abdessalem et al., 2018). This method randomly samples two parent candidate seeds from the current seed queue of size $N$ into a tournament, and the winner is chosen based on their objective function values. We repeat this process $k$ times to select $k$ winners. The winners are then paired up to serve as the selected parents for the following crossover and mutation steps. Then we parse each paired winner into an AST. Using ASTs, we introduce the LCST-based mutation strategy that can effectively mutate test inputs while keeping the input structure valid. The LCST problem is to find a common subtree (based on a bijective mapping) with the maximum number of nodes (Akutsu et al., 2015).

**Algorithm 2:** LCST-based Mutation

**Input :** the test input $A$, the grammar $G$, the test input $B$
**Output:** mutated test input $m_A$, mutated test input $m_B$

1 $\mathbb{P} \leftarrow \emptyset$;
2 parse $A$ according to G into an AST $A_{tree}$
3 parse $B$ according to G into an AST $B_{tree}$
4 $T_A \leftarrow$ Copy($A_{tree}$);
5 $T_B \leftarrow$ Copy($B_{tree}$);
6 $res\_tree \leftarrow$ LCST($A_{tree}, B_{tree}$);
7 **while** $res\_tree$ is not empty **do**
8 $\quad \mathbb{P} \leftarrow \mathbb{P} \cup \{res\_tree\}$;
9 $\quad T_A, T_B \leftarrow$ TrimTree($T_A, T_B, res\_tree$);
10 $\quad res\_tree \leftarrow$ LCST($T_A, T_B$);
11 **end**
12 **foreach** subtree $s \in \mathbb{P}$ **do**
13 $\quad A_{tree}, B_{tree} \leftarrow$ CrossMutation($A_{tree}, B_{tree}, s$);
14 **end**
15 $m_A, m_B \leftarrow$ ConvertFile($A_{tree}, B_{tree}$);

Algorithm 2 shows the procedure of our tree-based mutation. It takes as inputs test scenario description files A, B, and the grammar G. It first parses test inputs A and B according to G into ASTs (Line 2–3). After copying the ASTs (Line 4–5), it computes the largest common subtree of the two ASTs. Then it stores each obtained largest common subtree res_tree in the set $\mathbb{P}$, and trims $T_A$ and $T_B$ according to the res_tree structure until $T_A$ and $T_B$ have no largest common subtree (Line 7–11). Next, for each subtree s in $\mathbb{P}$, $A_{tree}$ and $B_{tree}$ are individually matched with the structure of s, and subsequently, the variable leaf nodes of the matched structures in $A_{tree}$ and $B_{tree}$ are paired (Line 12–14). For each pair of leaf nodes matched by the two ASTs, the algorithm adopts the simulated binary crossover (SBX) (Deb and Agrawal, 1995) to simulate the working principle of the single-point crossover operator on binary strings. It sets a suitable distribution index $\eta_d$ and probability $p_c$ to promote the diversity of the test inputs. Further, it applies the polynomial mutation (Deb and Deb, 2014) to each value for leaf nodes with a pre-specified mutation rate $m_r$ and the mutation magnitude $\eta_m$ to promote mutations. Finally, it converts ASTs to mutated scenario description files $m_A$ and $m_B$ (Line 15).

### 3.4. Guidance criteria design

For conventional software programs, various structural code coverage is often employed as the guidance criteria in the fuzzing process. Different from traditional software, modern ADS contain multiple deep learning algorithms and models, which rely on a large amount of labeled data to predict or classify new input data after training. The particularity of ADS construction makes the traditional code coverage method ineffective, which is confirmed in many studies of testing DNN models and DNN-based software (Chen et al., 2020; Feng et al., 2020). To overcome this challenge and ensure testing efficiency, we incorporate failure-inducing and diversity-inducing properties, then design testing criteria as guidance in the test generation process.

For the failure-inducing property, we design an objective $F_{failure}$ to measure inconsistent behaviors during a simulation and maximize it. Given a concrete scenario $x$, the objective can be denoted as follows:

$$F_{failure}(x) = IB(S, \mathbb{I})/N_f \tag{2}$$

where $N_f$ represents the total number of video frames we collected from concrete scenario $x$, $IB(S, \mathbb{I})$ represents the number of inconsistent behaviors of a concrete autonomous driving scenario, which is detailed in Section 4.3.

For the diversity-inducing property, we take an information-theoretic perspective and develop an entropy-based objective function. This objective function assigns more energy to seeds that promote diversity. To further discover new error categories, we attempt to fuzz different seeds in each iteration. Next, we present more details of our design.

In the fuzzing campaign, we reserve a $L_s \times N_f$ probability matrix $\boldsymbol{P} = (p_{ij})$ for each iteration, which can be represented as:

$$\boldsymbol{P} = \begin{pmatrix} p_{11} & \cdots & p_{1N_f} \\ \vdots & \ddots & \vdots \\ p_{L_s1} & \cdots & p_{L_sN_f} \end{pmatrix} \tag{3}$$

where $L_s$ is the total number of logical scenarios, then we compute entropy $H(t)$ of the current iteration $t$ based on Shannon's entropy (Shannon, 1948) as follows:

$$H(t) = -\sum_{i=1}^{L_s} \sum_{j=1}^{N_f} p_{ij} \log\left(p_{ij}\right) \tag{4}$$

where $p_{ij}$ is the probability that failure belongs to the $j$th frame of $i$th logical scenario. To calculate the conditional information entropy after simulating scenario $x$ in the current state $t$, we define $H(x|t)$, which can be computed as:

$$H(x|t) = -\sum_{i=1}^{L_s} \sum_{j=1}^{N_f} p'_{ij} \log\left(p'_{ij}\right) \tag{5}$$

In the above equation, $p'_{ij}$ is the changed probability that failure belongs to the $j$th frame of $i$th logical scenario. We estimate entropy $H$ based on how often we have seen each error category. Therefore, we calculate a $L_s \times N_f$ total error frequency matrix $Y = (y_{ij})$ for each iteration as follows:

$$\boldsymbol{Y} = \begin{pmatrix} y_{11} & \cdots & y_{1N_f} \\ \vdots & \ddots & \vdots \\ y_{L_s1} & \cdots & y_{L_sN_f} \end{pmatrix} \tag{6}$$

To ensure $p_{ij} \neq 0$, the value of each element in the matrix is set to one before iteration takes place. During the iteration process, if the $j$th frame of the $i$th logical scenario experiences an error, the frequency value of the corresponding position is incremented by one. Then an estimator of the error discovery probability $p_{ij}$ is $\hat{p}_{ij} = y_{ij}/total_{error}$, where $y_{ij}$ the frequency of failures, $total_{error}$ is the sum of failure frequencies in $Y$ when iterating up to $t$ rounds.

Similar to $p_{ij}$, an estimator of $p'_{ij}$ is $\hat{p}'_{ij} = y'_{ij}/total'_{error}$. Note $y'_{ij} = y_{ij} + 1$ when a specific instance $x$ of $i$th logical scenario occurs an error at the $j$th frame. Compared to $total_{error}$, $total'_{error}$ additionally includes the errors found in the concrete scenario $x$.

By substituting $\hat{p}_{ij}$ and $\hat{p}'_{ij}$ into Eqs. (4) and (5) respectively, we can obtain the estimated entropy $\hat{H}_E(t)$ and $\hat{H}_E(t|x)$. Using these estimates, we are capable of calculating the information gain for $x$ at iteration $t$, i.e. $Gain(t, x) = \hat{H}_E(x|t) - \hat{H}_E(t)$. Additionally, we track the number of times each seed has been fuzzed to select different logical scenarios. Finally, we express the diversity objective $F_{div}$ for a concrete scenario $x$ as:

$$F_{div}(x) = \mathcal{T}\left(Gain(t, x)\right) + \frac{1}{e^{a*choose+b}} \tag{7}$$

where $\mathcal{T}$ normalizes the information gain values of the seeds in the seed queue to the range $[0, 1]$. $a$, $b$ are coefficients, and $choose$ represents the number of times the seed has been fuzzed. Putting the above objectives together, we obtain the following criteria that our fuzzer attempts to optimize:

$$F(x) = c_{failure}F_{failure}(x) + c_{div}F_{div}(x) \tag{8}$$

where $c_{failure}$ and $c_{div}$ are coefficients that are used to weigh the impact of the failure and diversity objectives, respectively. After performing generation operations in each iteration, the fuzzer updates the testing criteria for each seed in the queue.

### 3.5. Test oracle construction

A complete ADS must include braking, acceleration, and steering controllers. Testers usually assess the steering performance based on collision avoidance and riding comfort. During testing, it is crucial to consider behaviors that may not lead to a collision but can impact riding comfort (Burkhard et al., 2018). Therefore, one of the significant challenges in testing a single controller is to create the system's specifications manually, as it essentially involves recreating the logic of human drivers (Tian et al., 2018; Zhang et al., 2018c; Zhou et al., 2020). To avoid this issue, we leverage metamorphic relations (Chen et al., 2018) between the car behaviors across the original image and its transformed image. The core idea is that we can define relationships between the vehicle's behaviors across certain types of transformations. For instance, the autonomous car's steering angle should not change hugely for the same image under any semantic transformation. The violation of MRs often reveals potential defects.

Our approach is designed to realize the semantic MRs for ADS. Specifically, we denote the steering controllers of autonomous driving as $\mathbb{S}$, which predicts the car's steering behaviors. The MRs can be defined as given the original image stream $\mathbb{I}$, and semantic transformations $\gamma \in \mathbb{T}$ can generate various mutation images from each image $i \in \mathbb{I}$. The semantic MR to test $\mathbb{S}$ with additional transformed images can be formalized as:

$$\forall i \in \mathbb{I} \wedge \forall \gamma \in \mathbb{T}, \zeta\{\mathbb{S}[\gamma(i)], \mathbb{S}[i]\} \tag{9}$$

where $\zeta$ is a criterion asserting the equality of $\mathbb{S}$ steering outputs. The given MR is defined such that regardless of how the image is synthesized by applying semantic transformation $\gamma$ on $i$, the steering angle is expected to be consistent with that of the original image.

However, asserting the equality of $\mathbb{S}$ outputs is too strict, as there is usually no single correct steering angle for a given image. This approach may result in many false positives. In order to make the MRs more permissive, we allow variations within the error ranges of the original input scenarios. We define the number of inconsistent behaviors of ADS in Section 4.3. Given semantic MRs, we can simply check the test oracle whether it is satisfied in the testing process of $\mathbb{S}$.

## 4. Experiment design

In this section, we introduce our experimental setup, including the dataset and steering models under test, parameter settings, evaluation metrics, and the research questions we study. To conduct the experiments, we implement the fuzzing framework upon TensorFlow 2.5.0 and use CARLA 0.9.13 as the simulator. All experiments are performed on a Ubuntu 21.10 desktop with GeForce RTX 3070, one 16-core processor with 3.80 GHz and 32 GB physical memory.

FuzzScene is designed to systematically test the autonomous driving system via semantic-guided fuzzing. To this end, we empirically explore the following four research questions (RQ):

- **RQ1:** Can FuzzScene generate test suites triggering more faults in comparison to baselines?
- **RQ2:** How diverse are the faults generated by FuzzScene?
- **RQ3:** Can we leverage erroneous behaviors found by FuzzScene to enhance the steering controllers?

### 4.1. Dataset and steering controllers

***Dataset.*** For both training and testing datasets, we capture and collect a large number of video frames using the camera mounted on the ego vehicle from multiple CARLA route maps. The corresponding ground truth steering angles are acquired from the control signals, where negative steering angles indicate left turns, and positive values denote right turns. The data collected and generated in this paper has a maximum steering angle of ±25 degrees. To ensure that the predicted steering angles fall within the range of −1 and 1, the steering angle is scaled by a factor of 1/25.

***Controllers.*** To evaluate the performance of our techniques effectively, we leverage four popular steering controller models, which have been widely used in autonomous driving testing (Pei et al., 2017; Tian et al., 2018). Specifically, we test three models based on the Dave self-driving car architecture from NVIDIA with different configurations, denoted as Dave-orig (NVIDIA, 2016), Dave-norminit (Gildenblat, 2016), Dave-dropout (Staravoitau, 2016), as well as the Epoch model from the Udacity challenge (Udacity, 2016):

- **Dave-orig** fully replicates the original architecture presented in NVIDIA's Dave system.
- **Dave-norminit** removes the first batch normalization layer and normalizes the initialized network weights.
- **Dave-dropout** is another publicly available steering model that modifies the original Dave model by cutting down the number of convolutional layers and fully connected layers and inserting dropout layers among the fully connected layers.
- **Epoch** model consists of a single convolutional neural network that achieves a high rank in the Udacity self-driving challenge.

In order to determine whether FuzzScene is an effective tool for testing ADS, we obtain pre-trained steering models by training them with the collected image frames. Note that in original testing scenarios, Dave-orig, Dave-normint, Dave-dropout, and Epoch achieved 0.017, 0.016, 0.028, and 0.026 MSE, respectively, which indicate they performed well.

### 4.2. Scenario and parameter settings

***Scenario Select.*** We select six logical scenarios (i.e. $L_s = 6$) from various CARLA route maps, including Town04, Town06, and Town07. These logical scenarios inspired by the NHTSA report mainly describe the ego car following a leading vehicle. To ensure safe driving on the road, the ego car needs to make correct steering decisions for each scene. We capture corresponding frames when FuzzScene runs a concrete scenario, and we capture enough video frames ($N_f = 125$) in our experiment design.

***Parameter Settings.*** FuzzScene focuses on finding inconsistent behaviors of ADS by creating realistic semantic transformations. For the learning-to-rank sampling process, the hyper-parameters $gen_m$ and $q_r$ are set to 12 and 4. We set the selection times k for each generation to 4. In the crossover & mutation step, we configure $\eta_d$, $p_c$, $m_r$, and $\eta_m$ to be 10, 0.5, 0.05, and 5, respectively, to promote diversity in the offspring. We set the default values for $c_{failure}$, $c_{div}$, $a, b$ in the objective function (defined in Section 3.4) to 1, 1, 1, 0. The default values control the contribution of the failure-inducing property ($c_{failure}$) and the diversity-inducing property ($c_{div}$). The positive signs of $c_{failure}$ and $c_{div}$ signify the aim to maximize $F_{failure}$ and $F_{div}$. The fuzzing settings include 25 generations with four simulations per generation in the experiment, and the maximum size of the seed queue is 50.

### 4.3. Metric and baseline comparison

***Metric.*** We employ the MSE to measure the testing performance of FuzzScene and answer research questions in this paper. And we express it as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{\theta}_{gi} - \theta_{mi}\right)^2 \tag{10}$$

where $\hat{\theta}_{gi}$ represents the ground truth steering label of frame $i$, $\theta_{mi}$ is the respective output predicted by the steering model and $n$ is the number of frames.

In this research, an autonomous driving system is defined to act consistently if its steering angle prediction falls within certain error bounds after modifying the semantic information of driving scenarios (Zhang et al., 2018c). We represent the number of inconsistent behaviors of a concrete autonomous driving scenario as follows:

$$IB(S, \mathbb{I}) = \sum_{i \in \mathbb{I}} f\left((\mathbb{S}[i] - \mathbb{S}[\gamma(i)])^2 > \lambda MSE_{orig}\right) \tag{11}$$

where $\mathbb{S}$ denotes the autonomous driving model, and $\mathbb{I}$ is the image stream of the driving scenario, $i$ denotes the $i$th frame in $\mathbb{I}$. $\gamma$ denotes the transformer which can change the semantic information of the original image. $f$ is an indicator function that outputs 1 or 0 if and only if the input is true or false, $MSE_{orig}$ is the mean squared error of the original scenario, and $\lambda$ is a configurable parameter set to be 1 in experiments.

In order to reflect the diversity of faults found by FuzzScene, we use the concept of error category. The total error categories of logical scenario $l \in \mathbb{L}$ we found are defined as:

$$Error\_Category(l) = \sum_{i=1}^{N_l} \mathbb{1}\left[\exists x_l \text{ s.t. } IB(\mathbb{S}, i) = 1\right] \tag{12}$$

where $x_l$ is a concrete scenario instantiated from logical scenario $l$, $N_l$ is the number of scenes (frames) instantiated by $l$, and $\mathbb{1}$ is a mapping of a condition truth value to a numeric value in {0, 1}.

***Baseline.*** We noticed that several research works are proposed to generate tests for ADS. However, unlike these works, our technique handles diverse logical scenarios during the fuzzing process to test the steering controller component and satisfies the metamorphic relations when running scenarios.

- To evaluate the effectiveness of FuzzScene in finding erroneous behaviors, we use three baselines: random search (RANDOM), random sampling (RD-SAM), and the fuzz algorithm with objective $F_{div}$ (FA-DIV). The RD-SAM method generates the initial seeds randomly, and the FA-DIV method runs the fuzzer without $F_{failure}$ in the objective function.
- To assess the ability of FuzzScene to detect diverse faults, we leverage the fuzz algorithm with $F_{failure}$ (FA-FAIL) as the baseline. The FA-FAIL method runs the fuzzer without $F_{div}$ in the objective function.

## 5. Results, analysis and discussion

This section shows the experiment result and then analyzes the performance of our approach.

### 5.1. Answer to RQ1

To evaluate the fault detection ability of FuzzScene, we explore whether it can find more realistic inconsistent behaviors. We compare it with the three baselines described in Section 4.3. Note that all the inconsistent behaviors are generated by valid semantic scenarios, as they are transformed using our grammar-aware generation strategy.

**Results.** Fig. 4 shows the number of inconsistent behaviors found by the four methods for each controller. FuzzScene found 31%, 9%, 31%, and 29% more misbehaviors than the best baseline method under each configuration, respectively. As the iteration number increases, the FuzzScene framework discovers significantly more inconsistent behaviors than the RANDOM method.

To further prove the effectiveness of FuzzScene in detecting inconsistent behaviors, we present six examples of such behaviors intercepted by FuzzScene, as illustrated in Fig. 3. These inconsistencies are discovered across various CARLA route maps (Town04, Town06, and Town07). Although only minor semantic modifications are added to the original scene, the steering angle predicted by the controller model is changed substantially.

**Discussion.** After analysis, our technique is capable of identifying various combinations of transformations, which represent different environmental semantics (e.g., rain intensity, vehicle color, etc.) and lead to potential failures, according to Fig. 3. Additionally, Fig. 4 shows that FuzzScene, FA-DIV, and RD-SAM find more inconsistent behaviors than the RANDOM method for all configurations, thereby indicating that our designed fuzzing framework is a productive approach for identifying bugs by navigating large search spaces. Moreover, the proposed learning-to-rank sampling and $F_{failure}$ are effective as the number of detected defects increased substantially after implementing the failure-inducing objective and learning-to-rank sampling strategy.

### 5.2. Answer to RQ2

As failure-inducing inputs usually are very dense and close to each another (Chen et al., 2004a,b), similar faults may reflect the same defect. Therefore, to comprehensively analyze the steering controller, we aim to detect more inconsistent behaviors and find diverse faults. In RQ2, we analyze the diversity of faults generated by FuzzScene, i.e., whether generated test scenarios can trigger distinct error categories. We compare FuzzScene with FA-FAIL using the error category metric described in Section 4.3. Further, we classify each error according to the logical scenario and calculate the Gini impurity (Feng et al., 2020) metric to show the impurity of inconsistent behaviors.

**Results.** Fig. 5 presents the number of error categories detected by FuzzScene and FA-FAIL. FuzzScene found 48.2%, 82.1%, 44.6%, and 64.8% more error categories than the baseline method for each configuration, respectively. Table 2 further demonstrates the diversity of generated scenes measured by Gini impurity. We can observe that FuzzScene with both $F_{div}$ and $F_{failure}$ objective functions as test generation guidance can trigger more error categories and achieve higher Gini impurity for all settings.

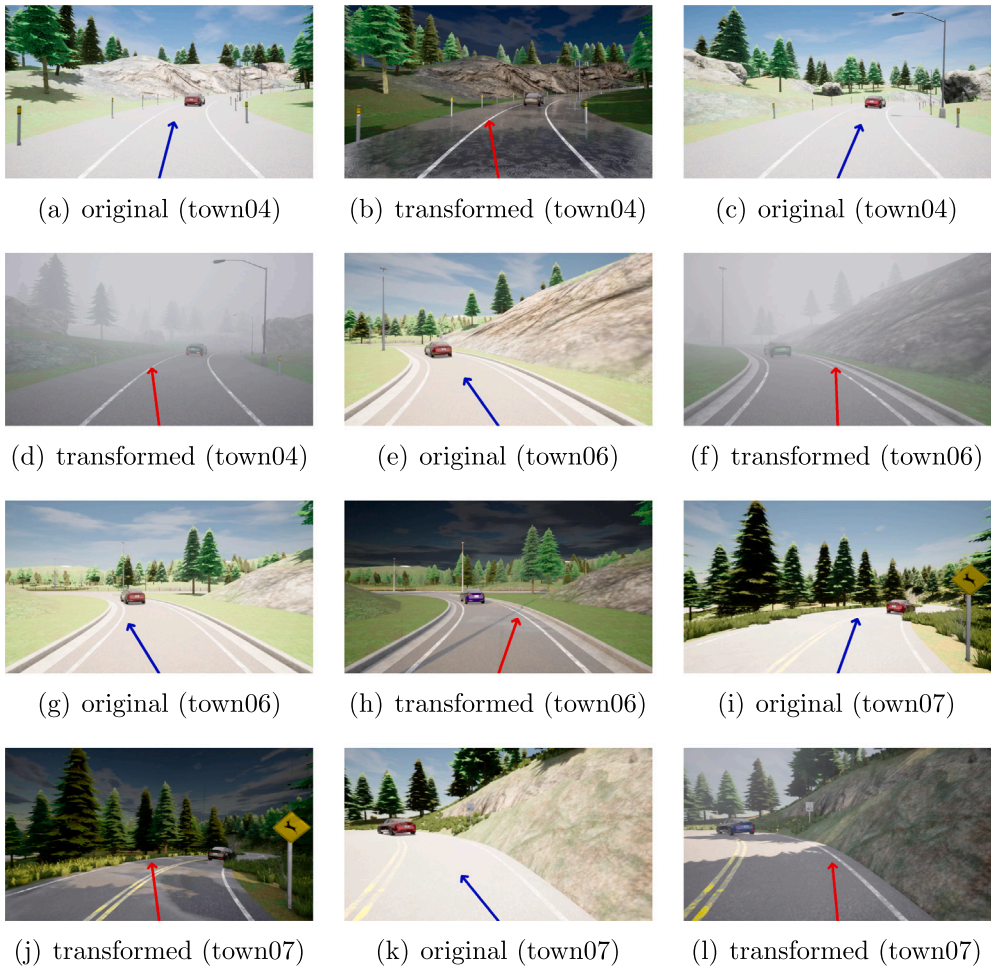


**Fig. 3.** Visualization samples showing inconsistent behaviors detected by FuzzScene using transformed scenes. For original scenes the arrows are marked in blue, while for the transformed scenes they are marked in red.

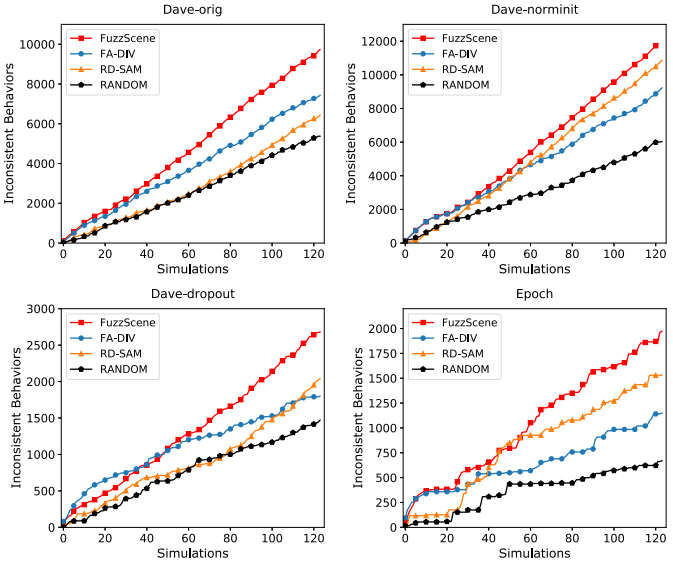


**Fig. 4.** The number of inconsistent behaviors found on the steering controllers over simulations.

**Discussion.** We calculate the number of triggering error categories and the impurity of the inconsistent behaviors to show the diversity of faults found by FuzzScene. Intuitively, the Gini value reflects the probability that two samples randomly selected from erroneous behaviors belong to different logical scenarios. Therefore, a higher Gini value indicates greater impurity of the faults, i.e., the more diverse the inconsistent behaviors. Furthermore, considering errors belonging to specific categories are unlikely to be generated, even with an exhaustive search, the theoretical maximum number of error categories (i.e., 750) cannot always be achieved. Despite this limitation, our framework is able to trigger a large number of error categories and achieve high Gini values. The experimental results show that the proposed diversity-inducing objective, $F_{div}$, is effective in increasing the fault diversity.

**Table 2**
Error category impurity measured by the Gini.

| Controller | FA-FAIL | FuzzScene |
|---|---|---|
| Dave-orig | 0.19 | **0.83** |
| Dave-norminit | 0.60 | **0.80** |
| Dave-dropout | 0.25 | **0.82** |
| Epoch | 0.30 | **0.82** |

### 5.3. Answer to RQ3

Here we examine whether retraining the steering controllers with the inconsistent behaviors generated by FuzzScene helps improve their performance. To reduce the time cost associated with retraining, we randomly select 1800 detected inconsistent behaviors by FuzzScene in RQ2 for each controller and split the corresponding failure frames into

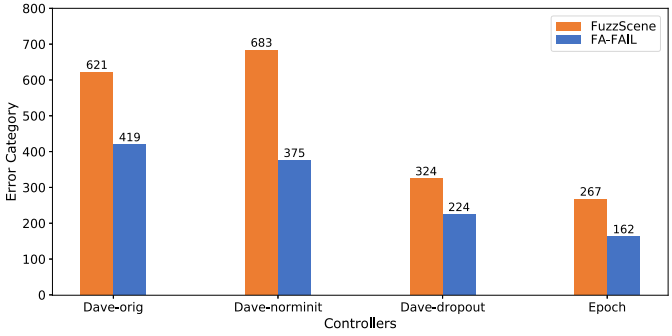


**Fig. 5.** The number of error categories found by fuzzers with different configurations over simulations.

**Table 3**
The test result comparison after retraining the controller with the transformed data generated by FuzzScene.

| Controller | Original MSE | Retrained MSE | Fixed scenes |
|---|---|---|---|
| Dave-orig | 0.097 | 0.031 | 580/900 |
| Dave-norminit | 0.192 | 0.027 | 577/900 |
| Dave-dropout | 0.080 | 0.026 | 882/900 |
| Epoch | 0.138 | 0.052 | 851/900 |

900 for retraining and 900 for testing. Subsequently, we evaluate the retrained steering model on the held-out 900 previously failing scenes. To assess the effectiveness of FuzzScene in generating superior tests, we perform a comparative analysis between FuzzScene retraining and the random retraining approach. We use the inconsistent behaviors revealed by FuzzScene during RQ2 and those revealed by the RANDOM method as training sets. We perform 40 simulations by randomly sampling from the parameter space and employ the generated data as a test set.

**Results.** Table 3 demonstrates the optimization effect and the number of fixed scenes for all controllers after retraining with the inconsistent behaviors generated by FuzzScene. We compute the MSE decrease and scene repair rate for each controller. The MSE reduction percentages for Dave-orig, Dave-norminit, Dave-dropout, and Epoch are 68%, 86%, 68%, and 62% respectively. The repair rates for Dave-orig, Dave-normint, Dave-dropout, and Epoch are 64%, 64%, 98%, and 95% respectively. Table 4 demonstrates the optimization impact resulting from the utilization of FuzzScene and random retraining techniques for all controllers. The performance improvements achieved by FuzzScene are significant when compared to random method retraining. Specifically, FuzzScene enhances the performance by 8.8% on Dave-orig, 47.5% on Dave-norminit, 25.8% on Dave-dropout, and 6.9% on Epoch models.

**Discussion.** Our experimental results show that the retrained controller succeeds in at least 64% of the originally failing scenes. Therefore, using a small number of erroneous behaviors generated by FuzzScene for retraining can significantly improve the performance of the controller model on the car-following task. Furthermore, the degree of improvement is correlated with the original steering precision. The experimental results show that relatively high steering precision may limit the room for improvement of the controller model.

### 5.4. Threats to validity

***Data Simulation.*** One of the primary threats to validity is data simulation. Our approach captures and collects the driving scenes from the open-source simulator for autonomous driving research. The simulated driving scenes may differ visually and physically from the real world because of the gap that exists between testing in the physical world and testing in a simulator. To alleviate this threat, we use the popular high-fidelity CARLA (Dosovitskiy et al., 2017) simulator, which has been built for flexibility and realism in both rendering and physics simulation. On the other hand, our virtual image generation technology has the potential to support industrial applications. For instance, we can test the autonomous driving system by directly injecting generated virtual image data after applying IPG CarMaker (Tests of camera, 2024).

**Table 4**
The MSE of the controllers after retraining using data generated by FuzzScene and RANDOM respectively.

| Controller | RANDOM retrain | FuzzScene retrain |
|---|---|---|
| Dave-orig | 0.034 | **0.031** |
| Dave-norminit | 0.061 | **0.032** |
| Dave-dropout | 0.031 | **0.023** |
| Epoch | 0.029 | **0.027** |

***The Driving Task Selection.*** In this paper, we limit our evaluation to the car-following task in testing autonomous driving vehicles due to the complexity of its applicable environments. Other driving behaviors, including cut-in and cut-out, etc., may also lead to erroneous decisions of the ADS. It is difficult to ensure the efficacy of FuzzScene in improving safety for every driving task. However, as car-following (Zhang et al., 2018a) is one of the most fundamental behaviors in autonomous driving, our fuzzing framework shows promising potential for use in testing other driving behaviors as well.

***Parameter Settings.*** Another threat could be caused by the configurable parameters, as the hyper-parameters (e.g., the coefficients of the objective functions) are not fully fine-tuned. Nevertheless, even with the current parameters, our approach has already outperformed the baselines with four steering models (Dave-orig, Dave-norminit, Dave-dropout, and Epoch). By fine-tuning the hyper-parameters, the performance of our proposed method can be further optimized.

## 6. Related work

In this section, we primarily introduce the related works on two aspects: testing approaches of autonomous driving systems and metamorphic testing.

### 6.1. Testing of autonomous driving systems

Automatically constructing and exploring diverse and challenging driving scenes is of great significance for the virtual testing of autonomous driving systems, due to the numerous possible traffic scenarios and their complexity (Hauer et al., 2019; Mullins et al., 2018; Tang et al., 2023). A recent set of approaches (Tian et al., 2018; Zhang et al., 2018c; Guo et al., 2022) leverages metamorphic testing techniques to examine inputs that trigger inconsistencies between the original and transformed driving scenes. Zhou and Sun (2019) employ metamorphic testing combined with fuzzing techniques to detect fatal errors in the LiDAR obstacle perception system of the Baidu Apollo self-driving software. Han and Zhou (2020) then test the robustness of the self-driving vehicle in the face of unexpected situations and use metamorphic relations as a filtering tool to distinguish between genuine failures and false alarms. Additionally, existing accelerated evaluation methods concentrate on optimizing virtual testing and improving efficiency (Zhong et al., 2022a; Masuda et al., 2018). Gambi et al. (2019) generate effective simulation-based tests automatically by combining a genetic algorithm and procedural content generation and expose many safety-critical failures related to lane-keeping functionality of autonomous vehicles. Li et al. (2020) propose an automated fuzzing framework that can generate safety violation scenarios for self-driving cars. Luo et al. (2021) study the ADS test case prioritization techniques

and employ multi-objective search algorithms to find different combinations of requirements violations. Kim et al. (2022) present DriveFuzz, an end-to-end fuzzing framework designed to find bugs in ADSes that are readily exploitable by attackers.

The key differences between the related work mentioned above and FuzzScene are twofold: (1) The framework of FuzzScene is highly flexible and extensible. By simply modifying the variable leaf nodes parsed from the ASTs, we can adjust scene semantics like the color and body type of the car. In comparison, other approaches (Tian et al., 2018; Zhang et al., 2018c) require more extensive efforts to identify scene semantics, such as the car body type, from a real scene and to implement scene mutation; and (2) Test case diversity is essential for software testing (Chen et al., 2010), we design a guidance criterion to generate test suites with sufficient diversity and fault detection ability in FuzzScene. In contrast, existing methods (Gambi et al., 2019; Li et al., 2020; Luo et al., 2021) focus solely on safety-related guidance, leading to the generation of redundant scenarios with similar motion misbehavior of the ego vehicle.

### 6.2. Fuzz testing

Fuzz testing (a.k.a. fuzzing) is an automatic and effective software testing technique that can discover both correctness and security bugs (Liang et al., 2018). During the fuzzing process, the basic way to reveal hidden bugs is to repeatedly produce numerous suitable test cases for target programs and monitor the program's exceptions. Fuzz testing has been used to detect vulnerabilities in both traditional software (Li et al., 2018) and deep learning applications (Guo et al., 2018), and it tends to work well with relatively simple input formats such as audio or image. For more complex input formats such as protocol messages (Hu et al., 2018) or cloud service APIs (Atlidakis et al., 2020), researchers usually employ grammar-based fuzzing (Eberlein et al., 2020) to obey domain-specific constraints and narrow down the search space for generating effective and valid inputs.

In this paper, we develop a semantic-guided fuzzing framework for the virtual testing of ADS. Specifically, our approach fuzzes the constrained input space by leveraging the grammar-aware generation strategy. In addition, we design a testing guidance criterion to accelerate the search for safety-critical driving scenarios.

## 7. Conclusion

In this paper, we present FuzzScene, a semantic-guided fuzzing technique for finding inconsistent behaviors of ADS during virtual testing. FuzzScene leverages the driving scenario description language OpenSCENARIO to describe scenarios. It innovatively designs a grammar-aware generation strategy and performs a guidance criterion for efficiently finding combinations of semantic transformations that result in more failures. FuzzScene has been evaluated on four steering controllers. The experiment results demonstrate that the transformed data generated by FuzzScene can effectively detect diverse inconsistent behaviors and improve controller model performance via retraining.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

We would like to thank anonymous reviewers for their insightful and constructive comments. This research was partially funded by the National Natural Science Foundation of China under Grant No. 61932012, and the Science, Technology and Innovation Commission of Shenzhen Municipality, China (No. CJGJZD20200617103001003 and No. No.2021Szvup057).

## References

Abdessalem, R.B., Nejati, S., Briand, L.C., Stifter, T., 2018. Testing vision-based control systems using learnable evolutionary algorithms. In: Chaudron, M., Crnkovic, I., Chechik, M., Harman, M. (Eds.), Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018. ACM, pp. 1016–1026. http://dx.doi.org/10.1145/3180155.3180160.

Abeysirigoonawardena, Y., Shkurti, F., Dudek, G., 2019. Generating adversarial driving scenarios in high-fidelity simulators. In: International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019. IEEE, pp. 8271–8277. http://dx.doi.org/10.1109/ICRA.2019.8793740.

Akutsu, T., Tamura, T., Melkman, A.A., Takasu, A., 2015. On the complexity of finding a largest common subtree of bounded degree. Theoret. Comput. Sci. 590, 2–16.

Althoff, M., Urban, S., Koschi, M., 2018. Automatic conversion of road networks from opendrive to lanelets. In: 2018 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI), Singpapore, Singapore, July 31 - Aug. 2, 2018. IEEE, pp. 157–162. http://dx.doi.org/10.1109/SOLI.2018.8476801.

2020. ASAM. https://www.asam.net/standards/detail/openscenario.

Atlidakis, V., Geambasu, R., Godefroid, P., Polishchuk, M., Ray, B., 2020. Pythia: Grammar-based fuzzing of REST APIs with coverage-guided feedback and learning-based mutations. CoRR abs/2005.11498. arXiv:2005.11498.

Bagschik, G., Menzel, T., Maurer, M., 2018. Ontology based scene creation for the development of automated vehicles. In: 2018 IEEE Intelligent Vehicles Symposium, IV 2018, Changshu, Suzhou, China, June 26-30, 2018. IEEE, pp. 1813–1820. http://dx.doi.org/10.1109/IVS.2018.8500632.

2022. Baidu's self-driving taxis now a common sight in Beijing. https://asia.nikkei.com/Business/China-tech/Baidu-s-self-driving-taxis-now-a-common-sight-in-Beijing.

Ben Abdessalem, R., Nejati, S., Briand, L.C., Stifter, T., 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. pp. 63–74.

Bender, P., Ziegler, J., Stiller, C., 2014. Lanelets: Efficient map representation for autonomous driving. In: 2014 IEEE Intelligent Vehicles Symposium Proceedings, Dearborn, MI, USA, June 8-11, 2014. IEEE, pp. 420–425. http://dx.doi.org/10.1109/IVS.2014.6856487.

Braiek, H.B., Khomh, F., 2020. On testing machine learning programs. J. Syst. Softw. 164, 110542.

Burkhard, G., Vos, S., Munzinger, N., Enders, E., Schramm, D., 2018. Requirements on driving dynamics in autonomous driving with regard to motion and comfort. In: 18. Internationales Stuttgarter Symposium: Automobil-Und Motorentechnik. Springer, pp. 683–697.

Chao, Q., Bi, H., Li, W., Mao, T., Wang, Z., Lin, M.C., Deng, Z., 2020. A survey on visual traffic simulation: Models, evaluations, and applications in autonomous driving. Comput. Graph. Forum 39 (1), 287–308. http://dx.doi.org/10.1111/cgf.13803.

Chen, T.Y., Kuo, F., Liu, H., Poon, P., Towey, D., Tse, T.H., Zhou, Z.Q., 2018. Metamorphic testing: A review of challenges and opportunities. ACM Comput. Surv. 51 (1), 4:1–4:27. http://dx.doi.org/10.1145/3143561.

Chen, T.Y., Kuo, F.-C., Merkel, R.G., Tse, T., 2010. Adaptive random testing: The art of test case diversity. J. Syst. Softw. 83 (1), 60–66.

Chen, T.Y., Leung, H., Mak, I.K., 2004a. Adaptive random testing. In: Maher, M.J. (Ed.), Advances in Computer Science - ASIAN 2004, Higher-Level Decision Making, 9th Asian Computing Science Conference, Dedicated To Jean-Louis Lassez on the Occasion of His 5th Cycle Birthday, Chiang Mai, Thailand, December 8-10, 2004, Proceedings. In: Lecture Notes in Computer Science, vol. 3321, Springer, pp. 320–329. http://dx.doi.org/10.1007/978-3-540-30502-6_23.

Chen, T., Merkel, R., Wong, P., Eddy, G., 2004b. Adaptive random testing through dynamic partitioning. In: Fourth International Conference OnQuality Software, 2004. QSIC 2004. Proceedings. pp. 79–86. http://dx.doi.org/10.1109/QSIC.2004.1357947.

Chen, J., Yan, M., Wang, Z., Kang, Y., Wu, Z., 2020. Deep neural network test coverage: How far are we? arXiv preprint arXiv:2010.04946.

de Gelder, E., Paardekooper, J., 2017. Assessment of automated driving systems using real-life scenarios. In: IEEE Intelligent Vehicles Symposium, IV 2017, Los Angeles, CA, USA, June 11-14, 2017. IEEE, pp. 589–594. http://dx.doi.org/10.1109/IVS.2017.7995782.

Deb, K., Agrawal, R.B., 1995. Simulated binary crossover for continuous search space. Complex Systems 9 (2).

Deb, K., Deb, D., 2014. Analysing mutation schemes for real-parameter genetic algorithms. Int. J. Artif. Intell. Soft Comput. 4 (1), 1–28. http://dx.doi.org/10.1504/IJAISC.2014.059280.

Diaz-Diaz, A., Ocaña, M., Llamazares, A., Huélamo, C.G., Revenga, P.A., Bergasa, L.M., 2022. HD maps: Exploiting opendrive potential for path planning and map monitoring. In: 2022 IEEE Intelligent Vehicles Symposium, IV 2022, Aachen, Germany, June 4-9, 2022. IEEE, pp. 1211–1217. http://dx.doi.org/10.1109/IV51971.2022.9827297.

Ding, W., Chen, B., Li, B., Eun, K.J., Zhao, D., 2021. Multimodal safety-critical scenarios generation for decision-making algorithms evaluation. IEEE Robot. Autom. Lett. 6 (2), 1551–1558.

Dosovitskiy, A., Ros, G., Codevilla, F., López, A.M., Koltun, V., 2017. CARLA: an open urban driving simulator. In: 1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings. In: Proceedings of Machine Learning Research, vol. 78, PMLR, pp. 1–16.

Eberlein, M., Noller, Y., Vogel, T., Grunske, L., 2020. Evolutionary grammar-based fuzzing. In: Aleti, A., Panichella, A. (Eds.), Search-Based Software Engineering - 12th International Symposium, SSBSE 2020, Bari, Italy, October 7-8, 2020, Proceedings. In: Lecture Notes in Computer Science, vol. 12420, Springer, pp. 105–120. http://dx.doi.org/10.1007/978-3-030-59762-7_8.

Feng, Y., Shi, Q., Gao, X., Wan, J., Fang, C., Chen, Z., 2020. Deepgini: prioritizing massive tests to enhance the robustness of deep neural networks. In: Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. pp. 177–188.

Fremont, D.J., Dreossi, T., Ghosh, S., Yue, X., Sangiovanni-Vincentelli, A.L., Seshia, S.A., 2019. Scenic: a language for scenario specification and scene generation. In: McKinley, K.S., Fisher, K. (Eds.), Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019. ACM, pp. 63–78. http://dx.doi.org/10.1145/3314221.3314633.

Gambi, A., Müller, M., Fraser, G., 2019. Automatically testing self-driving cars with search-based procedural content generation. In: Zhang, D., Møller, A. (Eds.), Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, Beijing, China, July 15-19, 2019. ACM, pp. 318–328. http://dx.doi.org/10.1145/3293882.3330566.

Gangopadhyay, B., Khastgir, S., Dey, S., Dasgupta, P., Montana, G., Jennings, P.A., 2019. Identification of test cases for automated driving systems using Bayesian optimization. In: 2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019, Auckland, New Zealand, October 27-30, 2019. IEEE, pp. 1961–1967. http://dx.doi.org/10.1109/ITSC.2019.8917103.

Gao, F., Duan, J., He, Y., Wang, Z., 2019. A test scenario automatic generation strategy for intelligent driving systems. Math. Probl. Eng. 2019, http://dx.doi.org/10.1155/2019/3737486.

Gildenblat, J., 2016. Visualizations for understanding the regressed wheel steering angle for self driving cars. https://github.com/jacobgil/keras-steering-angle-visualizations.

Guo, A., Feng, Y., Chen, Z., 2022. LiRTest: augmenting LiDAR point clouds for automated testing of autonomous driving systems. In: Ryu, S., Smaragdakis, Y. (Eds.), ISSTA '22: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, South Korea, July 18 - 22, 2022. ACM, pp. 480–492. http://dx.doi.org/10.1145/3533767.3534397.

Guo, J., Jiang, Y., Zhao, Y., Chen, Q., Sun, J., 2018. Dlfuzz: Differential fuzzing testing of deep learning systems. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 739–743.

Haklay, M.M., Weber, P., 2008. OpenStreetMap: User-generated street maps. IEEE Perv. Comput. 7 (4), 12–18. http://dx.doi.org/10.1109/MPRV.2008.80.

Han, J.C., Zhou, Z.Q., 2020. Metamorphic fuzz testing of autonomous vehicles. In: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops. pp. 380–385.

Hauer, F., Schmidt, T., Holzmüller, B., Pretschner, A., 2019. Did we test all scenarios for automated and autonomous driving systems? In: 2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019, Auckland, New Zealand, October 27-30, 2019. IEEE, pp. 2950–2955. http://dx.doi.org/10.1109/ITSC.2019.8917326.

Hu, Z., Shi, J., Huang, Y., Xiong, J., Bu, X., 2018. Ganfuzz: a GAN-based industrial network protocol fuzzing framework. In: Kaeli, D.R., Pericàs, M. (Eds.), Proceedings of the 15th ACM International Conference on Computing Frontiers, CF 2018, Ischia, Italy, May 08-10, 2018. ACM, pp. 138–145. http://dx.doi.org/10.1145/3203217.3203241.

2022. Japan approves use of highly autonomous 'level-4' self-driving vehicles from april'23. https://www.japantimes.co.jp/news/2022/12/20/business/level-3-highly-autonomous-self-driving-vehicles/.

Junietz, P., Wachenfeld, W., Klonecki, K., Winner, H., 2018. Evaluation of different approaches to address safety validation of automated driving. In: Zhang, W., Bayen, A.M., Medina, J.J.S., Barth, M.J. (Eds.), 21st International Conference on Intelligent Transportation Systems, ITSC 2018, Maui, HI, USA, November 4-7, 2018. IEEE, pp. 491–496. http://dx.doi.org/10.1109/ITSC.2018.8569959.

Kalra, N., Paddock, S.M., 2016. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? Transp. Res. A 94, 182–193.

Kaur, P., Taghavi, S., Tian, Z., Shi, W., 2021. A survey on simulators for testing self-driving cars. In: 2021 Fourth International Conference on Connected and Autonomous Driving. MetroCAD, IEEE, pp. 62–70. http://dx.doi.org/10.1109/MetroCAD51599.2021.00018.

Kim, S., Liu, M., Rhee, J.J., Jeon, Y., Kwon, Y., Kim, C.H., 2022. Drivefuzz: Discovering autonomous driving bugs through driving quality-guided fuzzing. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 1753–1767.

Li, G., Li, Y., Jha, S., Tsai, T., Sullivan, M.B., Hari, S.K.S., Kalbarczyk, Z., Iyer, R.K., 2020. AV-FUZZER: finding safety violations in autonomous driving systems. In: Vieira, M., Madeira, H., Antunes, N., Zheng, Z. (Eds.), 31st IEEE International Symposium on Software Reliability Engineering, ISSRE 2020, Coimbra, Portugal, October 12-15, 2020. IEEE, pp. 25–36. http://dx.doi.org/10.1109/ISSRE5003.2020.00012.

Li, W., Pan, C., Zhang, R., Ren, J., Ma, Y., Fang, J., Yan, F., Geng, Q., Huang, X., Gong, H., Xu, W., Wang, G., Manocha, D., Yang, R., 2019. AADS: augmented autonomous driving simulation using data-driven algorithms. Sci. Robot. 4 (28), http://dx.doi.org/10.1126/scirobotics.aaw0863.

Li, J., Zhao, B., Zhang, C., 2018. Fuzzing: a survey. Cybersecurity 1 (1), 6. http://dx.doi.org/10.1186/s42400-018-0002-y.

Liang, H., Pei, X., Jia, X., Shen, W., Zhang, J., 2018. Fuzzing: State of the art. IEEE Trans. Reliab. 67 (3), 1199–1218. http://dx.doi.org/10.1109/TR.2018.2834476.

Lou, G., Deng, Y., Zheng, X., Zhang, M., Zhang, T., 2022. Testing of autonomous driving systems: where are we and where should we go? In: Roychoudhury, A., Cadar, C., Kim, M. (Eds.), Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022. ACM, pp. 31–43. http://dx.doi.org/10.1145/3540250.3549111.

Luo, Y., Zhang, X.-Y., Arcaini, P., Jin, Z., Zhao, H., Ishikawa, F., Wu, R., Xie, T., 2021. Targeting requirements violations of autonomous driving systems by dynamic evolutionary search. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 279–291.

Majumdar, R., Mathur, A.S., Pirron, M., Stegner, L., Zufferey, D., 2019. Paracosm: A language and tool for testing autonomous driving systems. CoRR abs/1902.01084. arXiv:1902.01084.

Masuda, S., Nakamura, H., Kajitani, K., 2018. Rule-based searching for collision test cases of autonomous vehicles simulation. IET Intell. Transp. Syst. 12 (9), 1088–1095. http://dx.doi.org/10.1049/iet-its.2018.5335.

2020. Measurable scenario description language reference. https://www.foretellix.com/wp-content/uploads/2020/07/M-SDL_LRM_OS.pdf.

Miller, B.P., Zhang, M., Heymann, E.R., 2022. The relevance of classic fuzz testing: Have we solved this one? IEEE Trans. Softw. Eng. 48 (6), 2028–2039. http://dx.doi.org/10.1109/TSE.2020.3047766.

Mullins, G.E., Stankiewicz, P.G., Hawthorne, R.C., Gupta, S.K., 2018. Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles. J. Syst. Softw. 137, 197–215.

NVIDIA, 2016. Nvidia-autopilot-keras. https://github.com/0bserver07/Nvidia-Autopilot-Keras.

Scenario-Based Testing with OpenSCENARIO. https://www.vector.com/int/en/know-how/virtual-test-drives/openscenario/.

PEGASUS, 2019. Scenario description. URL https://www.pegasusprojekt.de/files/tmpl/PDF-Symposium/04_Scenario-Description.pdf.

Pei, K., Cao, Y., Yang, J., Jana, S., 2017. DeepXplore: Automated whitebox testing of deep learning systems. In: Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017. ACM, pp. 1–18. http://dx.doi.org/10.1145/3132747.3132785.

Queiroz, R., Berger, T., Czarnecki, K., 2019. GeoScenario: An open DSL for autonomous driving scenario representation. In: 2019 IEEE Intelligent Vehicles Symposium, IV 2019, Paris, France, June 9-12, 2019. IEEE, pp. 287–294. http://dx.doi.org/10.1109/IVS.2019.8814107.

Rong, G., Shin, B.H., Tabatabaee, H., Lu, Q., Lemke, S., Mozeiko, M., Boise, E., Uhm, G., Gerow, M., Mehta, S., Agafonov, E., Kim, T.H., Sterner, E., Ushiroda, K., Reyes, M., Zelenkovsky, D., Kim, S., 2020. LGSVL simulator: A high fidelity simulator for autonomous driving. In: 23rd IEEE International Conference on Intelligent Transportation Systems, ITSC 2020, Rhodes, Greece, September 20-23, 2020. IEEE, pp. 1–6. http://dx.doi.org/10.1109/ITSC45102.2020.9294422.

Schöner, H.-P., 2018. Simulation in development and testing of autonomous vehicles. In: 18. Internationales Stuttgarter Symposium. Springer, pp. 1083–1095. http://dx.doi.org/10.1007/978-3-658-21194-3_82.

Shannon, C.E., 1948. A mathematical theory of communication. Bell Syst. Tech. J. 27 (3), 379–423. http://dx.doi.org/10.1002/j.1538-7305.1948.tb01338.x.

Staravoitau, A., 2016. Behavioral cloning: end-to-end learning for self-driving cars. https://github.com/navoshta/behavioral-cloning.

Sun, J., Zhang, H., Zhou, H., Yu, R., Tian, Y., 2022. Scenario-based test automation for highly automated vehicles: A review and paving the way for systematic safety assurance. IEEE Trans. Intell. Transp. Syst. 23 (9), 14088–14103. http://dx.doi.org/10.1109/TITS.2021.3136353.

Tang, S., Zhang, Z., Zhang, Y., Zhou, J., Guo, Y., Liu, S., Guo, S., Li, Y.-F., Ma, L., Xue, Y., et al., 2023. A survey on automated driving system testing: Landscapes and trends. ACM Trans. Softw. Eng. Methodol..

2024. Tests of camera-based systems made easy through direct injection of image data. https://ipg-automotive.com/en/products-solutions/hardware/video-interface-box-x/.

Tian, Y., Pei, K., Jana, S., Ray, B., 2018. DeepTest: automated testing of deep-neural-network-driven autonomous cars. In: Chaudron, M., Crnkovic, I., Chechik, M., Harman, M. (Eds.), Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018. ACM, pp. 303–314. http://dx.doi.org/10.1145/3180155.3180220.

Udacity, 2016. Using deep learning to predict steering angles. https://medium.com/udacity/challenge-2-using-deep-learning-to-predict-steering-angles-f42004a36ff3.

Ulbrich, S., Menzel, T., Reschka, A., Schuldt, F., Maurer, M., 2015. Defining and substantiating the terms scene, situation, and scenario for automated driving. In: IEEE 18th International Conference on Intelligent Transportation Systems, ITSC 2015, Gran Canaria, Spain, September 15-18, 2015. IEEE, pp. 982–988. http://dx.doi.org/10.1109/ITSC.2015.164.

2022. US releases new driver-assist crash data, and surprise, it's mostly tesla. https://www.theverge.com/2022/6/15/23168088/nhtsa-adas-self-driving-crash-data-tesla.

Wang, Y., Yu, R., Qiu, S., Sun, J., Farah, H., 2022. Safety performance boundary identification of highly automated vehicles: A surrogate model-based gradient descent searching approach. IEEE Trans. Intell. Transp. Syst. http://dx.doi.org/10.1109/TITS.2022.3191088.

Zhang, X., Khastgir, S., Jennings, P.A., 2020. Scenario description language for automated driving systems: A two level abstraction approach. In: 2020 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2020, Toronto, on, Canada, October 11-14, 2020. IEEE, pp. 973–980. http://dx.doi.org/10.1109/SMC42975.2020.9283417.

Zhang, Y., Lin, Q., Wang, J., Verwer, S., Dolan, J.M., 2018a. Lane-change intention estimation for car-following control in autonomous driving. IEEE Trans. Intell. Veh. 3 (3), 276–286. http://dx.doi.org/10.1109/TIV.2018.2843178.

Zhang, S., Peng, H., Zhao, D., Tseng, H.E., 2018b. Accelerated evaluation of autonomous vehicles in the lane change scenario based on subset simulation technique. In: Zhang, W., Bayen, A.M., Medina, J.J.S., Barth, M.J. (Eds.), 21st International Conference on Intelligent Transportation Systems, ITSC 2018, Maui, HI, USA, November 4-7, 2018. IEEE, pp. 3935–3940. http://dx.doi.org/10.1109/ITSC.2018.8569800.

Zhang, M., Zhang, Y., Zhang, L., Liu, C., Khurshid, S., 2018c. DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. pp. 132–142.

Zhao, X., Robu, V., Flynn, D., Salako, K., Strigini, L., 2019. Assessing the safety and reliability of autonomous vehicles from road testing. In: Wolter, K., Schieferdecker, I., Gallina, B., Cukier, M., Natella, R., Ivaki, N.R., Laranjeiro, N. (Eds.), 30th IEEE International Symposium on Software Reliability Engineering, ISSRE 2019, Berlin, Germany, October 28-31, 2019. IEEE, pp. 13–23. http://dx.doi.org/10.1109/ISSRE.2019.00012.

Zhong, Z., Hu, Z., Guo, S., Zhang, X., Zhong, Z., Ray, B., 2022a. Detecting multi-sensor fusion errors in advanced driver-assistance systems. In: Ryu, S., Smaragdakis, Y. (Eds.), ISSTA '22: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, South Korea, July 18 - 22, 2022. ACM, pp. 493–505. http://dx.doi.org/10.1145/3533767.3534223.

Zhong, Z., Kaiser, G., Ray, B., 2022b. Neural network guided evolutionary fuzzing for finding traffic violations of autonomous vehicles. IEEE Trans. Softw. Eng..

Zhou, H., Li, W., Kong, Z., Guo, J., Zhang, Y., Yu, B., Zhang, L., Liu, C., 2020. DeepBillboard: systematic physical-world testing of autonomous driving systems. In: Rothermel, G., Bae, D. (Eds.), ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020. ACM, pp. 347–358. http://dx.doi.org/10.1145/3377811.3380422.

Zhou, Z.Q., Sun, L., 2019. Metamorphic testing of driverless cars. Commun. ACM 62 (3), 61–67. http://dx.doi.org/10.1145/3241979.