SimADFuzz: Simulation-Feedback Fuzz Testing for Autonomous Driving Systems

HUIWEN YANG, Nanjing University of Aeronautics and Astronautics, China
YU ZHOU*, Nanjing University of Aeronautics and Astronautics, China
TAOLUE CHEN*, Birkbeck, University of London, United Kingdom

Autonomous driving systems (ADS) have achieved remarkable progress in recent years. However, ensuring their safety and reliability remains a critical challenge due to the complexity and uncertainty of driving scenarios. In this paper, we focus on simulation testing for ADS, where generating diverse and effective testing scenarios is a central task. Existing fuzz testing methods face limitations, such as overlooking the temporal and spatial dynamics of scenarios and failing to leverage simulation feedback (*e.g.,* speed, acceleration and heading) to guide scenario selection and mutation. To address these issues, we propose SimADFuzz, a novel framework designed to generate high-quality scenarios that reveal violations in ADS behavior. Specifically, SimADFuzz employs violation prediction models, which evaluate the likelihood of ADS violations, to optimize scenario selection. Moreover, SimADFuzz proposes distance-guided mutation strategies to enhance interactions among vehicles in offspring scenarios, thereby triggering more edge-case behaviors of vehicles. Comprehensive experiments demonstrate that SimADFuzz outperforms state-of-the-art fuzzers by identifying 73 more unique violations, including 5 reproducible cases of vehicle-vehicle, vehicle-pedestrian and vehicle-roadside collisions. These results demonstrate SimADFuzz's effectiveness in enhancing the robustness and safety of autonomous driving systems.

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

Additional Key Words and Phrases: Autonomous Driving Systems, Fuzz Testing, Simulation-based Testing

ACM Reference Format:
Huiwen Yang, Yu Zhou, and Taolue Chen. 2024. SimADFuzz: Simulation-Feedback Fuzz Testing for Autonomous Driving Systems. *ACM Trans. Softw. Eng. Methodol.* 1, 1, Article 1 (January 2024), 32 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Autonomous driving systems (ADS), such as Apollo[1] and Autoware[2], have made significant progress in recent years. Various technologies have been developed to enable vehicles to operate autonomously without human intervention [7, 9]. However, ensuring the safety and reliability of these systems remains a critical challenge [21]. As of March 2024, the California Department of Motor

*Corresponding author.
[1]Apollo, https://github.com/ApolloAuto/apollo
[2]Autoware, https://github.com/autowarefoundation/autoware

Authors' Contact Information: Huiwen Yang, yhw_yagol@nuaa.edu.cn, Nanjing University of Aeronautics and Astronautics, Nanjing, China; Yu Zhou, zhouyu@nuaa.edu.cn, Nanjing University of Aeronautics and Astronautics, Nanjing, China; Taolue Chen, t.chen@bbk.ac.uk, Birkbeck, University of London, London, United Kingdom.

ACM 1557-7392/2024/1-ART1
https://doi.org/XXXXXXX.XXXXXXX

Vehicles reported 695 traffic accidents involving autonomous vehicles,[3] including 133 collisions in 2023. These incidents underscore the urgent need for comprehensive and effective testing of ADS before deployment.

Recent research has focused on leveraging various technologies to test the performance and reliability of ADS. For instance, adversarial attacks, widely used in computer vision, have been applied to test the robustness of ADS perception modules by exposing vulnerabilities in object detection and classification [59, 69]. Similarly, software testing techniques such as search-based testing [6] and fuzz testing [28, 30, 31, 68] have shown great potential in identifying defects and vulnerabilities in ADS. While these methods have achieved significant results, challenges remain in ensuring comprehensive coverage and scalability for real-world scenarios.

Simulation-based testing has emerged as a widely adopted method for evaluating ADS due to its efficiency and cost-effectiveness compared to real-road testing [45]. By generating diverse and realistic driving scenarios, simulation-based testing can evaluate ADS under various conditions, identifying potential violations such as collisions, unsafe lane changes, and traffic rule violations. This approach is indispensable for uncovering safety issues and improving the reliability of ADS.

The quality of simulation scenarios is critical to the effectiveness of simulation-based testing for ADS [13]. Various scenario generation methods, such as DriveFuzz [30] and Doppel [28], have been proposed. In general, these methods leverage genetic algorithms to generate offspring scenarios through selection, crossover, and mutation of parent scenarios. Scenarios are evaluated and prioritized using pre-designed fitness functions, often considering factors such as the behavior of the *ego vehicle*[4] or minimum distances between vehicles. Random mutation strategies are then applied to generate offspring scenarios. However, these methods face several limitations:

(1) *Limitations in scenario fitness evaluation.* Existing fitness functions typically prioritize scenarios based on simple aggregation methods, such as maximum, average, or median values. However, these approaches overlook the sequential nature of driving scenarios, which consist of discrete temporal scenes capturing dynamic interactions and behaviors. Simply aggregating attributes fails to account for these temporal dynamics, potentially leading to suboptimal prioritization.
(2) *Limitations in mutation strategies.* Scenarios involve a large number of mutable elements, resulting in an enormous search space. While random mutation is a natural approach, it fails to consider the interactions between mutable elements, such as vehicles and pedestrians. As a result, this strategy may struggle to produce high-quality scenarios that effectively challenge the ADS.

In this paper, we propose SimADFuzz, a simulation-feedback fuzz testing method for ADS to address the limitations of existing methods. SimADFuzz monitors and collects simulation feedback, including the coordinates and physical states of vehicles, during the simulation. Based on genetic algorithms, it generates high-quality testing scenarios. Unlike previous work that primarily uses feedback for scenario selection, SimADFuzz innovatively leverages feedback to extract temporal features for scenario fitness evaluation and to design effective mutation strategies.

To address the limitations of scenario fitness evaluation, SimADFuzz optimizes scenario selection using model-based fitness evaluation methods. Specifically, we propose a Violation Prediction Model (VPM), which integrates a Transformer [57] encoder to capture the continuous and dynamic nature of driving scenarios. It first embeds the driving scenario into a fixed-length vector to represent the high-dimensional feature space. Then, it predicts the probability of violations through

[3]DMV Autonomous Vehicle Collision Reports, https://www.dmv.ca.gov/portal/vehicle-industry-services/autonomous-vehicles/autonomous-vehicle-collision-reports/

[4]The vehicle controlled by ADS is referred to as the ego vehicle, while other vehicles are referred to as *NPC vehicles*.

a fully connected layer. Scenarios with higher predicted violation probabilities are prioritized for subsequent crossover and mutation operations to generate offspring scenarios for further testing.

Additionally, to overcome the limitations of mutation strategies, SimADFuzz adopts a distance-guided mutation strategy. This strategy dynamically adjusts the mutation probability of NPC vehicles based on their proximity to ego vehicles, increasing the likelihood of interactions. By prioritizing NPC vehicles closer to the ego vehicle for mutation, SimADFuzz generates offspring scenarios that are more likely to expose potential safety issues while maintaining scenario diversity.

We conduct extensive experiments on two ADS, *i.e.,* InterFuser [47] and LMDrive [46]. InterFuser is a top-tier agent that secured 2nd place on the CARLA leaderboard,[5] and LMDrive is the first research prototype ADS that leverages large language models for end-to-end autonomous driving. The results show that SimADFuzz detects 18 more unique violations than the baseline, which employ random selection and mutation strategies, in 3-hour fuzzing. Moreover, in 10-hour fuzzing, SimADFuzz discovers 150, 85, and 73 more violations than AV-Fuzzer [31], DriveFuzz [30], and TM-Fuzzer [34], respectively. Finally, we manually check 9 violations discovered by SimADFuzz, including 5 collision scenarios triggered by InterFuser and LMDrive. These violations are reproducible, demonstrating the effectiveness of SimADFuzz in generating safety-critical scenarios and detecting ADS violations.

The main contributions of this paper are summarized as follows:

(1) We propose a novel fuzz testing method for ADS, named SimADFuzz, which leverages simulation feedback to generate high-quality scenarios. SimADFuzz effectively discovers violations in ADS by dynamically analyzing vehicle states and interactions during simulation.
(2) We develop a model-based scenario fitness evaluation approach. By utilizing violation prediction models and incorporating a Transformer encoder, SimADFuzz captures the temporal features of driving scenarios, enabling more accurate prioritization of high-risk scenarios.
(3) We introduce distance-guided mutation strategies that mutate NPC vehicles based on their proximity to ego vehicles. This approach increases the likelihood of interactions, generating diverse and challenging scenarios that expose potential safety issues in ADS.
(4) We conduct extensive experiments to evaluate the effectiveness of SimADFuzz. Results demonstrate that SimADFuzz detects more violations compared to state-of-the-art methods, including collisions and lane invasion. To facilitate reproducibility and further research, we release the implementation publicly.[6]

Structure. The rest of this paper is organized as follows. Section 2 introduces the preliminaries of simulation-based testing and genetic algorithms. Section 3 illustrates the limitations of existing methods through motivating examples. Section 4 details the design and implementation of SimADFuzz. Section 5 presents the experimental results and discusses potential threats to validity. Section 6 reviews related work in the field. Finally, Section 7 concludes the paper and proposes directions for future research.

2 Preliminaries

Simulation-based testing is a widely adopted approach to evaluate the performance and reliability of ADS [17, 35, 65]. This method uses virtual scenarios as structured test cases to execute and validate target systems. Genetic algorithms [5, 39], inspired by natural selection, are commonly employed in fuzz testing to generate diverse and high-quality test cases. Several fuzz testing

[5]The 1st place agent, ReasonNet, has not made its code publicly available. Cf. CARLA leaderboard, https://leaderboard.carla.org/leaderboard/

[6]SimADFuzz, https://github.com/yagol2020/SimADFuzz

methods, such as AV-Fuzzer [31], DriveFuzz [30], Doppel [28], and TM-Fuzzer [34], have been proposed for simulation-based ADS testing.

Figure 1 illustrates the framework of simulation-based fuzz testing for ADS. This framework takes the ADS under test as input and produces violation reports as output. It consists of three main components: seed scenario generation, simulation, and genetic operators. First, seed scenarios are generated and executed in the simulation environment. During the execution, feedback information such as vehicle coordinates and velocities is collected from the simulator to evaluate the fitness of each scenario. Based on these fitness scores, the most promising scenarios are selected. These selected scenarios undergo crossover or mutation to produce offspring scenarios, which are then executed in the simulation environment for subsequent generations. During execution, the simulation environment detects various types of violations, such as collisions, unsafe lane changes, and traffic rule violations. This fuzzing process is repeated until the testing budget (*e.g.,* time) is exhausted. At the end of the process, a report summarizing the detected violations is generated to aid in evaluating the ADS's performance and identifying potential issues.

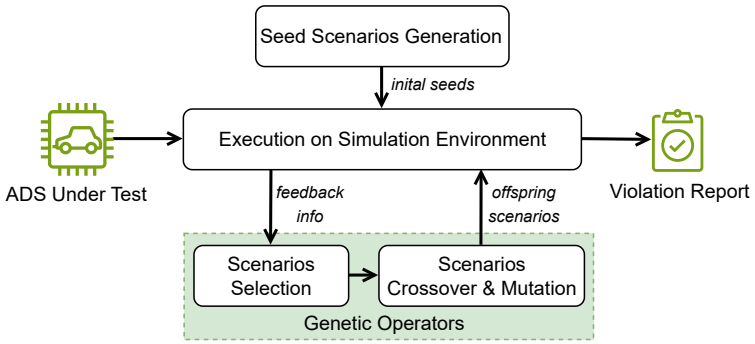


Fig. 1. Simulation-based Fuzz Testing Framework for ADS

In this framework, *scenario selection*, *crossover*, and *mutation* are the three main genetic operators that define the core of genetic algorithms. For scenario selection, genetic algorithms identify promising scenarios based on several metrics and evaluate their fitness scores using single- or multi-objective search methods. These evaluations are performed using fitness functions, which quantify the likelihood of a scenario causing violations [1]. Table 1 summarizes the commonly used metrics in simulation-based testing for ADS. As shown in Table 1, the fitness function evaluates scenarios to estimate their probability of triggering violations. For scenario crossover and mutation, random strategies are widely used. For instance, DriveFuzz [30] employs random mutation by altering weather conditions (*e.g.,* wind, cloud cover, and rain) or modifying pedestrian behaviors without utilizing any crossover operator. Similarly, Doppel [28] adopts random mutation by adding or removing traffic participants or modifying the starting and destination points of ego vehicles. On the other hand, Doppel's crossover operator swaps ego vehicles between two scenarios when their routes intersect.

In this paper, we aim to optimize two key genetic operators: *scenario selection* and *mutation*. The details of these optimizations are presented in Section 4.

Table 1. Fitness Metrics used in AV-Fuzzer, DriveFuzz, and Doppel

Method	Factor	Description
AV-Fuzzer[31]	d_{safe}	The maximum distance without colliding with other actors
	d_{stop}	The distance the vehicle will travel before coming to a complete stop
	Fitness score	$= d_{safe} - d_{stop}$
DriveFuzz[30]	ha	The times of hard acceleration
	hb	The times of hard braking
	ht	The times of hard turn
	os	The times of oversteer
	us	The times of understeer
	md	The minimum distances from ego vehicle to other actors
	Fitness score	$= -(ha + hb + ht + os + us - 1/md)$
Doppel[28]	$f_{min\_distance}$	The minimum distances from ego vehicle to other actors
	$f_{decision}$	The total number of unique decisions being made by all ego vehicles
	$f_{conflict}$	The total number of pairs of actors whose trajectory overlaps with another
	$f_{violation}$	The total number of violations across all ego vehicles
	Fitness score	Based on NSGA-2

3 Motivating Examples

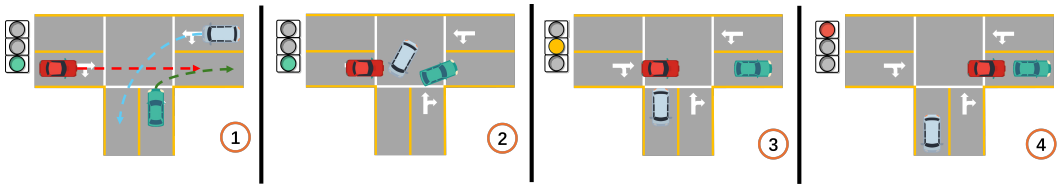
This section presents scenario examples to illustrate the limitations of existing methods and discusses the necessity of introducing model-based fitness evaluation and distance-guided mutation strategies.

As shown in Figure 2, the scenario involves three vehicles converging at a T-junction. The red vehicle intends to proceed straight, the white vehicle is making a left turn, and the green vehicle is turning right. The traffic light for the east-west direction is green, allowing all vehicles to proceed legally.

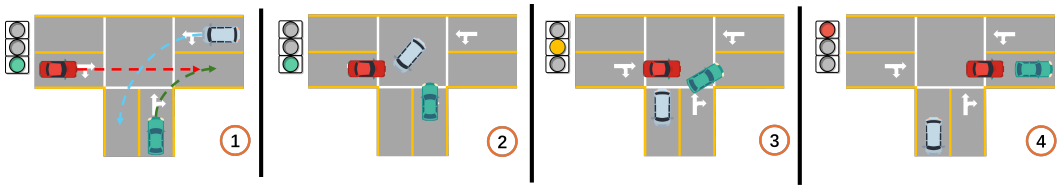
Figure 2a illustrates the scenario S_1. At time point 2, the white vehicle turns left and encounters the red vehicle travelling straight. To avoid a collision, the red vehicle brakes, maintaining a distance of 3 meters from the white vehicle. Meanwhile, the green vehicle completes its right turn quickly. At time point 3, as the traffic light changes to yellow, the red vehicle accelerates to pass through the junction. However, at time point 4, it brakes again due to the slow-moving green vehicle ahead, now maintaining a distance of 5 meters between them.

In a similar scenario S_2 depicted in Figure 2b, the key difference is that the start point of the green vehicle is further far from the T-junction. At time point 2, the red and white vehicles maintain a minimum distance of 3 meters, while the green vehicle just enters the junction. By time point 3, the red vehicle brakes once more to avoid a rear-end collision, with a distance of 4 meters. Finally, at time point 4, all three vehicles safely navigate through the T-junction without incident.

Table 2 presents the collected feedback information for the red vehicle, including the minimum distance to other vehicles and hard brake events during the simulation. Both DriveFuzz and Doppel incorporate the minimum distance (md) between vehicles as a key metric in their fitness functions. For example, DriveFuzz evaluates fitness based on the minimum distance and the number of hard braking (hb) events by the ego vehicle, while Doppel considers the minimum distance between all



(a) Scenario S_1: Green Vehicle Quickly Passes the T-junction without Interacting with the Red Vehicle



(b) Scenario S_2: Green Vehicle Turns Right and Causes the Red Vehicle to Brake at the T-junction

Fig. 2. Two Similar Scenarios of Multi-Vehicle Interactions at a T-junction

vehicles and integrates additional metrics using the NSGA-II algorithm [71]. However, both methods rely on aggregate functions, such as *minimum* or *count*, to evaluate the overall scenario fitness. As a result, DriveFuzz and Doppel assign identical fitness values to both scenarios ($S1_{md} = 3, S1_{hb} = 2$ and $S2_{md} = 3, S2_{hb} = 2$), thereby erroneously considering S_1 and S_2 to pose equivalent risks of triggering more violation behaviors.

Table 2. Feedback Information of the Red Vehicle during Simulation

Scenarios	Distance				Hard Brake			
	t_1	t_2	t_3	t_4	t_1	t_2	t_3	t_4
S_1	12	**3**	6	5	✗	✓	✗	✓
S_2	12	**3**	4	5	✗	✓	✓	✗

However, S_2 is considered riskier than S_1, as the red vehicle not only interacts with two different vehicles at one T-junction, but also triggers two hard braking events within a short period of time while continuously maintaining a close distance to other vehicles. These factors create a more complex and hazardous traffic situation compared to S_1. However, aggregate functions (*e.g.*, minimum) only capture the most severe moment (*e.g.*, the closest distance of 3 meters in S_2), overlooking other potential risks such as the 4-meters distance at time point 3 of S_2, which may indicate another collision risk.

Furthermore, DriveFuzz only counts the number of hard braking events without considering the spatial or temporal context of these events. For example, although the red vehicle triggers two hard braking events in both scenarios, the clustering of these events differs significantly. In scenario S_1, the two braking events occur in different road segments, allowing the ADS more time to react and adjust. In contrast, in scenario S_2 both events happen within a short time frame at the same T-junction, challenging the performance of the ADS and triggering more edge-case behaviors. Existing methods, however, reduce such events to simple numerical counts, failing to capture critical details that could indicate higher risk scenarios.

In summary, the complex scenario S_2 illustrated in Figure 2b should be prioritized for mutation to further explore potential risks. However, due to the reliance on insufficient aggregate functions

(*cf.* the first limitation), existing methods may assign the same or even lower fitness scores to such complex scenarios compared to simpler ones, leading to missed opportunities in identifying safety-critical behaviors of ADS.

In addition, dangerous or violating behaviors usually occur during interactions with other traffic participants. With more frequent interactions comes a higher risk of violations [41]. Consider another scenario shown in Figure 3, where the white vehicle gradually exits the conflict zone of the T-junction, increasing its distance from the other two vehicles. As the interactions decrease, it becomes less likely to exhibit further violating behaviors. Consequently, the white vehicle should be prioritized for mutation to increase interaction. However, both DriveFuzz and Doppel randomly select vehicles for mutation (*cf.* the second limitation), which inevitably results in redundant offspring scenarios.

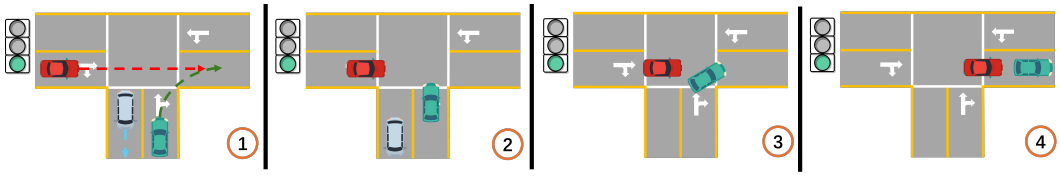


Fig. 3. Decreased Interaction Caused by the White Vehicle Driving Away From the Other Two Vehicles

To address these challenges, we propose SimADFuzz, a fuzz testing method that employs a Transformer encoder to effectively analyze the temporal dynamics within scenarios and evaluate the overall scenario using violation prediction models. Additionally, SimADFuzz integrates distance-guided mutation strategies to enhance the likelihood of interactions between vehicles, thereby improving the quality and relevance of offspring scenarios.

4 Approach

Figure 4 presents an overview of SimADFuzz, which comprises three modules: the simulation test engine, the simulation-feedback genetic algorithm, and the violation detector.

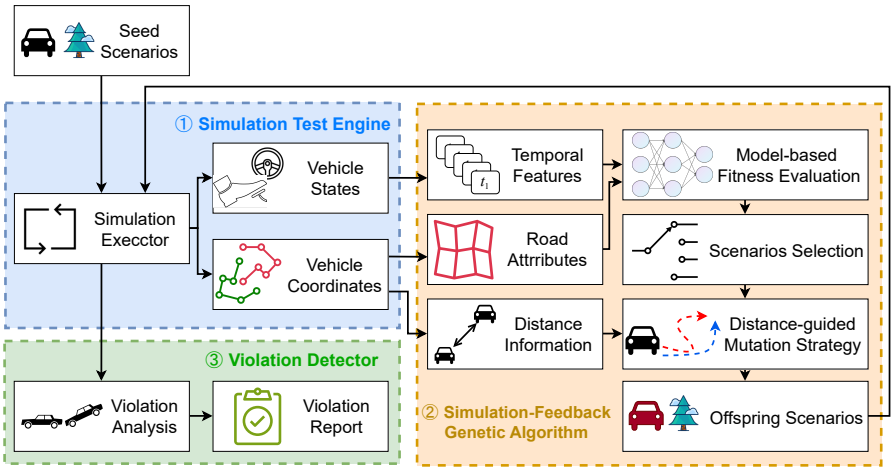


Fig. 4. Overview of SimADFuzz

First, SimADFuzz sends seed scenarios to the simulation test engine for execution. The simulation engine collects feedback information, including the coordinates and physical states of vehicles, as detailed in § 4.1.

Second, the simulation-feedback genetic algorithm processes the collected feedback information to extract temporal features and road attributes. It then evaluates the fitness score based on the violation prediction model and SDC-Scissor (described in § 4.2.1). The fitness scores guide the selection of scenarios, which are subjected to crossover (§ 4.2.2) and mutation (§ 4.2.3) to generate offspring scenarios that promote more interactions between vehicles and increase the likelihood of triggering violation behaviors.

Finally, the violation detector analyzes the interactions between vehicles, pedestrians, and traffic lights to identify five types of violations. Violation reports, along with the corresponding scenarios that can reproduce the violations, are output to assist ADS developers and testers in identifying defects, as detailed in § 4.3.

In the following section, we present the key components of SimADFuzz in detail.

4.1 Simulation Test Engine and Feedback Collection

To implement simulation-based testing, we design a simulation test engine that incorporates an autonomous vehicle simulator and a simulation-feedback collector.

The autonomous vehicle simulator is maintained by high-fidelity platforms (*e.g.,* CARLA [15], LGSVL [44]), or simulation tools designed for specific ADS (*e.g.,* Dreamview [2], AWSIM [55]). These simulators provide realistic environments, deploy vehicles and pedestrians at specific coordinates, and return sensor data such as RGB cameras, radar, GPS, and inertial measurement units.

Among these, CARLA and LGSVL have emerged as two widely used platforms in fuzz testing methods. However, LGSVL ceased maintenance and updates in 2022, limiting access to certain maps and assets. In contrast, CARLA has continuously improved its maps and API integrations, offering more comprehensive simulation capabilities that are well-suited for fuzz testing. Additionally, CARLA hosts official ADS performance benchmarks through the CARLA Leaderboard, making it a preferred choice for both research and industry. Consequently, SimADFuzz prioritizes CARLA as the primary simulation environment for generating driving scenarios, deploying ADS, and conducting fuzz testing.

The simulation-feedback collector gathers real-time feedback information by calling APIs provided by the simulator. Specifically, SimADFuzz collects two types of vehicle-related information, as shown in Table 3, *i.e.,* (1) vehicle coordinates, which indicate the vehicle's location and trajectory; and (2) physical states, which reflect the vehicle's behavior, including attributes such as speed and acceleration. This information captures the temporal features of vehicles during simulation and is used to evaluate scenario fitness through a model-based approach.

Table 3. Feedback Information collected by SimADFuzz

Type	Name	Description
Vehicle coordinates	loc_x	X-coordinate of the vehicle
	loc_y	Y-coordinate of the vehicle
Physical states	$speed_x$	X-direction speed of the vehicle
	$speed_y$	Y-direction speed of the vehicle
	acc_x	X-direction acceleration of the vehicle
	acc_y	Y-direction acceleration of the vehicle
	yaw	Heading of the vehicle

4.2 Simulation-Feedback Genetic Algorithm

Figure 5 illustrates the process of the simulation-feedback genetic algorithm. In SimADFuzz, the genetic representation (*i.e.,* chromosome) of a scenario used for simulation is composed of four parts. The green and red parts shown in Figure 5 represent the routes of the ego vehicle and NPC vehicles, respectively, defined by their start and end points. The yellow part represents the routes of pedestrians, while the blue part encodes weather conditions, such as rain or fog levels. Notably, all chromosomes maintain a fixed number of NPC vehicles and pedestrians throughout evolutionary operations. Specifically, when removing traffic participants during mutation, we replenish by generating different route for the corresponding parts, which can avoid excessive computational cost of generating too many traffic participants while maintaining realistic simulation during fuzz testing.

After the parent scenario completes simulation, SimADFuzz collects the coordinates and physical states of vehicles. These collected features are then used in a sequential process involving scenario selection, crossover, and mutation to generate offspring scenarios for the next generation of simulations.

We detail the key components of the simulation-feedback genetic algorithm, including model-based fitness evaluation, scenario selection, crossover strategies, and mutation strategies, in the following sections.

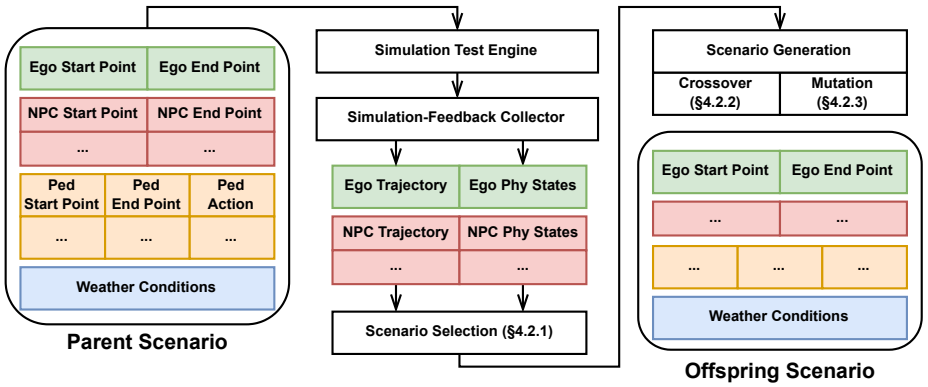


Fig. 5. Process of the Simulation-Feedback Genetic Algorithm

4.2.1 Model-based Fitness Evaluation and Scenario Selection. In general, a driving scenario is a sequence composed of several scenes, where each scene represents a snapshot of the simulation world [56]. Scenes may include actions, events, and other objects that characterize the driving environment. Based on this definition, we formalize a driving scenario S as a sequence $(s_1, \cdots, s_T)$ of T scenes with fixed time intervals. Each scene s_t is represented as $(v_1^t, \cdots, v_m^t)$, where m is the number of vehicles in S, and v_i^t refers to the features of vehicle v_i, including its coordinates and physical states at scene s_t.

However, these features only represent individual scenes within the scenario and fail to capture interactions between vehicles or behaviors such as turning or sudden braking. These interactions are critical for evaluating the potential of a scenario to trigger violations.

To address this limitation, we design the Violation Prediction Model (VPM) to evaluate the fitness score of driving scenarios by predicting the probability of triggering violations. Specifically, the VPM extracts temporal relationships and interactions among vehicles through the **driving scenario encoder**, which is a Transformer-based encoder embedding a driving scenario into a fixed-length vector representing the high-dimensional feature space. Then, the probability of violation (*i.e.,* the fitness score of the scenario) is predicted by the **violation prediction layer** and guides the selection of scenarios for further optimization and mutation. Details about the two main layers of VPM are described below.

Driving Scenario Encoder. The driving scenario encoder is designed to learn patterns from scene sequences using sequence models, which are effective in capturing relevant information from temporal data. Sequence models such as LSTM [25] and Transformer have been widely applied to tasks like driving behavior intention recognition [20] and trajectory/velocity prediction [19, 22, 37]. In this work, we adopt the Transformer for the driving scenario encoding task due to its superior performance in trajectory forecasting [23, 63]. In SimADFuzz, the Transformer encoder takes feedback information as input, represented as a tensor with the shape $T \times N_{info}$. Here, T denotes the number of scenes, and N_{info} represents the total amount of information collected for all vehicles in the scenario. Specifically, $N_{info} = 7 \times m$, where m is the number of vehicles, and the 7 features include coordinates, speed, acceleration, and heading. By leveraging the multi-head attention mechanism, the Transformer encoder embeds the scenario into a high-dimensional feature space, which is then used to predict the probability of violations.

Violation Prediction Layer. The violation prediction layer is designed to estimate the probability of violations, which serves as the fitness score for the scenario. It consists of a fully connected (FC) layer with a single output node, producing a scalar probability value. The FC layer combines and models relationships among features extracted from the entire scene sequence, capturing high-level interactions and temporal dependencies. Scenarios with higher predicted probabilities indicate a greater likelihood of causing violations in the ADS. Consequently, these scenarios are prioritized for selection in generating offspring scenarios during the optimization process.

The VPM evaluates scenarios and prioritizes them for crossover and mutation in the genetic algorithm. However, relying solely on a single fitness metric can be misleading, as it may fail to capture other critical characteristics of the scenarios being tested [16]. As shown in Table 1, existing works commonly evaluate scenarios using multiple fitness metrics to ensure a more comprehensive assessment. To address this limitation, we augment the SimADFuzz with three additional fitness metrics: minimum distance, number of unique violations [11, 28], and the score of SDC-Scissor (Self Driving Cars Cost Effective Test Selector; abbreviated as SDC-Score) [4].

- The minimum distance ($Dist_{\text{min}}$) is the smallest distance between the ego vehicle and other vehicles throughout the scenario simulation. A smaller distance typically indicates a higher likelihood of collisions or other violations, reflecting an increased level of risk in the scenario. Formally, $Dist_{\text{min}}$ is computed by Equation (1), where T is the time steps in the simulation, C denotes the NPC vehicles, e is the ego vehicle and $dist$ represents the Euclidean distance between two vehicles at time t.

$$Dist_{\text{min}} = \min_{t \in T, c \in C} dist(t, e, c) \tag{1}$$

- The number of unique violations (Num_{uvs}) refers to the number of violations triggered by the ego vehicle in the current scenario, after filtering out duplicates or highly similar violations. A higher count of unique violations suggests a more risky and complex scenario, potentially exposing more defects in the ADS by challenging its performance in diverse ways. Formally, Num_{uvs} is computed in Equation (2), where V denotes the violations detected in the current

Table 4. Name and Description of SDC-Features [4]

Name	Description
Direct distance	Euclidean distance between start and finish
Length	Total length of the driving path
Number L Turns	Number of left turns on the driving path
Number R Turns	Number of right turns on the driving path
Number Straight	Number of straight segments on the driving path
Total angle	Cumulative turn angle on the driving path
Median, Std, Max, Min, Mean angle	Median/Std/Maximum/Minimum/Average turn angle on the driving path
Median, Std, Max, Min, Mean radius	Median/Std/Maximum/Minimum/Average turn radius of the driving path
Full road diversity	The cumulative diversity of the full road composed of all segments
Mean road diversity	The mean diversity of the segments of a road

scenario, U is the repository of unique violations collected from all prior tests, *sim* evaluates the similarity between two violations. Note that newly identified unique violations are added to U after computation for future reference.

$$Num_{\text{uvs}} = \left|\left\{v \in V \,\middle|\, \forall u \in U,\ sim(v,u) < \theta\right\}\right| \tag{2}$$

- SDC-Scissor is a method that leverages machine learning models to identify and filter unlike scenarios to detect faults in ADS before executing them. It extracts static road attributes (*i.e.,* SDC-Features, as shown in Table 4) and sends them into trained models to classify scenarios as safe or unsafe. Complementarily, SimADFuzz focuses on capturing temporal features, such as dynamic interactions between vehicles over time. Combining these static and temporal features enhances the overall fitness evaluation by leveraging the strength of both approaches. The original SDC-Scissor treats scenario prediction as a binary classification problem based on the probability output from the model. We repurpose this probability as the SDC-Score (S_{sdc}) defined in Equation (3), where F_{sdc} denotes the feature vector representing SDC-Features.

$$S_{\text{sdc}} = model(F_{\text{sdc}}) \tag{3}$$

The use of the number of unique violations as a fitness metric is partially inspired by coverage-guided fuzz testing techniques [48, 64], which retain test cases that cover new basic blocks or branches and discard those that do not, thereby effectively exploring unseen code paths. As the coverage is determined relative to previously explored cases, these order-dependent metrics are useful to guide the search process toward unique crash or bugs that have not been discovered before. We define the number of unique violations in a similar manner: a violation is considered unique if it differs from previously observed violations in terms of the events' timestamp and spatial location within a given threshold. This allows us to identify and filter redundant scenarios that do not contribute to detecting new violations, thereby improving violations diversity. Although the same scenario may receive different fitness scores when evaluated at different time (due to prior violations), it aligns with the goal of encouraging diversity in violation-triggering behaviors. We remark that the number of unique violations as a fitness metric has also been successfully applied in ADS fuzzing studies [28, 34, 68], validating the effectiveness of this metric in guiding the search toward diverse and non-redundant violations.

To this end, we combine the VPM with three additional fitness metrics, treating scenario selection as a multi-objective optimization task. The goal is to identify solutions (*i.e.,* scenarios) that balance multiple objectives, such as increasing violation probabilities and risk exposure, or reducing the distance between vehicles. To achieve this, we leverage the Non-dominated Sorting Genetic Algorithm II (NSGA-2) [71], a widely used multi-objective optimization algorithm, to select scenarios based on the Pareto-optimal frontier. The reason behind our choice of NSGA-2 is twofold. NSGA-2 identifies a set of non-dominated solutions, meaning no other solutions perform better across all objectives. Through this mechanism, the most elite and effective genes (*e.g.,* the routes of NPC vehicles) are selected for crossover and mutation to facilitate more interactions with an ego vehicle. Moreover, by employing crowding-distance techniques, NSGA-2 ensures a diverse set of driving scenarios, effectively exploring the simulation search space while avoiding premature convergence to local optima.

4.2.2 Crossover Strategy. The crossover operator combines two parent scenarios to generate two offspring scenarios. In SimADFuzz, the crossover operation swaps the routes of NPC vehicles, pedestrians and weather conditions, while excluding the route of the ego vehicle to preserve interaction validity.

For the pedestrians, SimADFuzz randomly swaps half of the pedestrians between the two scenarios, increasing the diversity of pedestrians routes and behaviors.

For the NPC vehicles, SimADFuzz is designed to enhance the interaction likelihood between the ego vehicle and NPC vehicles. Specifically, for two parent scenarios S_1 and S_2, SimADFuzz checks whether the trajectory of an NPC vehicle NPC_i^{S1} in S_1 intersects with the trajectory of the ego vehicle EGO^{S2} in S_2. If it is the case, there might be a potential interaction between EGO^{S2} and NPC_i^{S1}, and SimADFuzz randomly selects one NPC vehicle from S_2 and swaps it with NPC_i^{S1}, generating two offspring scenarios.

For weather conditions, SimADFuzz swaps each individual weather parameters (*e.g.,* rain intensity, fog density, *etc.*) between the two parent scenarios with a default probability of 50%. This stochastic crossover operation enhances environmental diversity while maintaining weather parameters in valid ranges.

For the ego vehicle, its route is excluded from crossover to prevent invalidating NPC interactions. If the route of ego vehicle is modified, its trajectory would change in subsequent simulations, breaking the spatio-temporal alignment with NPC vehicles whose swapped trajectories are specifically designed to interact with the ego's unaltered path. Although altering the ego's route could explore diverse road structures, it decreases the probability of interaction with NPC vehicles. Therefore, SimADFuzz retains the ego's route to ensure NPC crossover validity.

4.2.3 Mutation Strategy. Mutation operators generate offspring scenarios based on a parent scenario. As shown in Figure 5, the chromosome of a scenario is categorized into three components: vehicles (including the ego and NPC vehicles), pedestrians, and weather. Among these, vehicles play a critical role in contributing to dynamic scenario variations. To enhance the likelihood of interactions with the ego vehicle, we focus on vehicle mutation and propose a distance-guided mutation strategy.

Distance-guided Mutation for Vehicles. Vehicles are essential traffic participants in creating dynamic scenarios for testing. As discussed in Section 3, *the probability of interactions and subsequent violations decreases as the distance between vehicles increases.* To address this, SimADFuzz employs a distance-guided mutation strategy to identify and remove NPC vehicles with low interaction potential. Specifically, two types of NPC vehicles are identified and removed:

- **Stuck Vehicles** refer to NPC vehicles that remain nearly stuck throughout the simulation, contributing minimally to scenario dynamics. Existing approaches primarily focus on identifying scenarios where the ego vehicle itself becomes stuck (*a.k.a.* stationary or paralysis) to identify ADS function failure [10, 30]. TM-Fuzzer [34] identifies and removes stationary NPC vehicles surrounding the ego vehicle to prevent it from getting in traffic congestion. Inspired by this, we focus on the effectiveness of NPC behaviors, identifying NPC vehicles that remain stationary throughout the simulation. For example, vehicles that are waiting for traffic lights over prolonged durations. Since these NPC vehicles are unlikely to interact meaningfully with the ego vehicle or challenge the ADS's edge-case handling capabilities, SimADFuzz removes them and generates new routes during the mutation process.
- **Leaving Vehicles** refer to NPC vehicles that are moving away from the ego vehicle, whose diminishing influence reduces their impact on the ADS perception module and decision-making processes. SimADFuzz therefore removes these departing vehicles and reassigns their routes.

The process of distance-guided mutation is detailed in Algorithm 1. The algorithm takes the following inputs: the set of NPC vehicles in the parent scenario (C), the ego vehicle in the parent scenario (e), the distance matrix (m_{dist}), a threshold for identifying stuck vehicles (w), and a time window size for identifying leaving vehicles (u). The algorithm outputs a modified set of NPC vehicles for the offspring scenario.

The m_{dist} is represented as a three-dimensional matrix with shape $T \times |V| \times |V|$, where T denotes the number of time steps, and $|V|$ represents the number of vehicles. Each element $m_{dist}(t, v_1, v_2)$ represents the Euclidean distance between vehicles v_1 and v_2 at timestamp t. The distance is computed as:

$$m_{dist}(t, v_1, v_2) = \begin{cases} \|v_1^t - v_2^t\|_2, & \text{if } v_1 \neq v_2 \\ 0, & \text{otherwise} \end{cases}$$

where $\|v_1^t - v_2^t\|_2$ denotes the Euclidean distance between the positions of vehicles v_1 and v_2 at time t.

Algorithm 1 begins by initializing an empty set D to record NPC vehicles marked for removal. Each vehicle $v \in C$ is evaluated based on two criteria:

1. **Stuck Vehicles**: The algorithm computes the total route length Δ_{route} for each vehicle v by summing the Euclidean distances between its positions at consecutive timestamps. If $\Delta_{route} < w$, indicating minimal movement throughout the simulation, the vehicle is added to D.

2. **Leaving Vehicles**: For each vehicle v, the algorithm computes the cumulative distance trend Δ_{dist} relative to the ego vehicle e over a sliding time window of size u. If Δ_{dist} remains positive (≥ 0) across all examined windows, the vehicle is marked as consistently moving away and added to D.

Vehicles in D are removed from the scenario. The function Mutate_Route modifies the routes of the remaining vehicles and the ego vehicle e, ensuring the mutated routes remain on the same road segment for contextual consistency. The function Add_Vehicles then replenishes the removed vehicles with new NPC vehicles, generating routes with randomly selected start and end points to explore new routes. The final mutated set of NPC vehicles is returned as part of the offspring scenario.

Figure 6 illustrates an example of distance-guided mutation. In the parent scenario (left), the red vehicle is identified as a *stuck vehicle*, as it remains stationary at a long red traffic light, while the yellow one is identified as a *leaving vehicle*, as its trajectory indicates it is consistently moving away from the ego vehicle (the blue vehicle), reducing the likelihood of interactions. These two vehicles are removed during the mutation process.

Algorithm 1: Distance-guided Mutation Process

Input : C - NPC vehicles in the parent scenario,
e - Ego vehicle in the parent scenario,
m_{dist} - Distance matrix,
w - Threshold for identifying stuck vehicles,
u - Time window size for identifying leaving vehicles
Output : C - Mutated NPC vehicles

$D \leftarrow \emptyset$;
for $v \in C$ **do**
  $\Delta_{route} \leftarrow \sum_{t=1}^{T} \|v^t - v^{t-1}\|_2$;
  if $\Delta_{route} < w$ **then**
    $D \leftarrow D \cup \{v\}$;
    continue;
  $flag \leftarrow True$;
  for $t \in [u, T]$ **do**
    $\Delta_{dist} \leftarrow \sum_{i=t-u}^{t}(m_{dist}(i+1, v, e) - m_{dist}(i, v, e))$;
    if $\Delta_{dist} < 0$ **then**
      $flag \leftarrow False$;
      break;
  if $flag$ **then**
    $D \leftarrow D \cup \{v\}$;
$C \leftarrow C \setminus D$;
MUTATE_ROUTE(C) ;
MUTATE_ROUTE(e) ;
ADD_VEHICLES(C, $|D|$) ;
return C;

In the offspring scenario (right), two new vehicles are added to replace the removed ones. One of the new vehicles intersects with the ego vehicle's trajectory at a roundabout, thereby enhancing the potential for interactions in the offspring scenario.

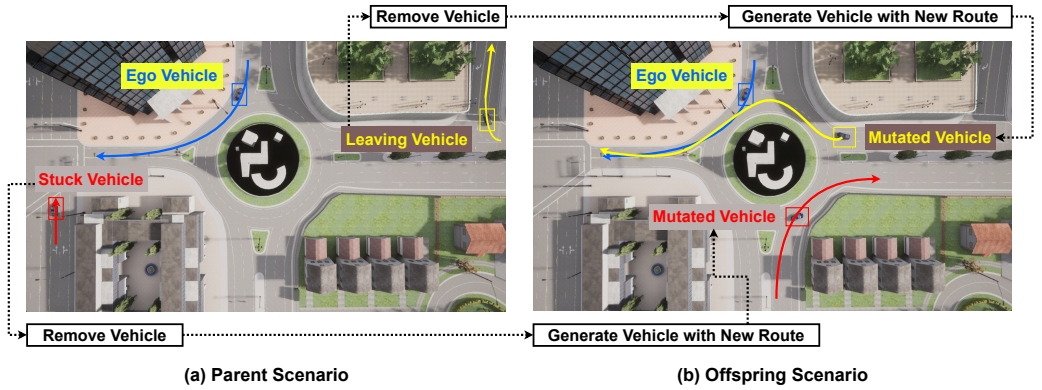


Fig. 6. An Example of Distance-guided Mutation

Mutation for Pedestrians and Weather Conditions. SimADFuzz also mutates pedestrians and weather conditions. Pedestrians are spawned near the ego vehicle to improve interaction likelihood, as their relatively low speed makes distant interactions unlikely. Weather conditions are randomly sampled from predefined ranges, such as rain intensity (0–100) and sun altitude (-90° to 90°), to explore diverse environmental scenarios.

4.3 Violation Detector and Reproduction

The violation detector module is responsible for identifying misbehavior and violations triggered by ego vehicles during simulation. SimADFuzz supports detecting the following violations:

Collision: Collisions are one of the most fundamental violations that an ADS must avoid. Collisions are detected when the ego vehicle comes into physical contact with other static or dynamic objects. This detector is implemented using CARLA's built-in collision sensors.

Lane Invasion: Lane invasions are detected when the ego vehicle crosses restricted road lanes, such as crossing solid lines that indicate non-crossable lane boundaries during lane changes. This detector is implemented using CARLA's built-in lane invasion sensors.

Speeding: Speeding violations are detected when the ego vehicle's speed consistently exceeds the specific road's speed limit for a duration $T_{speeding}$. This detector is implemented by analyzing the ego vehicle's historical speed data and comparing it against the speed limit.

Running Red Lights: Running red light violations are detected when the ego vehicle drives through an intersection during a red light. This detector is implemented by monitoring the traffic light state and the ego vehicle's coordinates within the intersection.

Stuck: Stuck behaviors often indicate ADS failures or system disablement but may also result from external factors such as traffic congestion. These violations are detected when the ego vehicle remains stationary beyond a predefined duration T_{stuck}. This detector is implemented by analyzing the ego vehicle's historical speed data and verifying whether the speed remains consistently zero over the duration T_{stuck}.

When SimADFuzz identifies a violation, it saves the entire scenario (including the states of the ego vehicle, NPC vehicles, pedestrians and weather conditions) to facilitate reproduction. Additionally, SimADFuzz utilizes CARLA's API (`Client.start_recorder`) to record scenarios with more detailed information (such as the states of traffic lights and vehicle dynamics). SimADFuzz supports replaying both types of scenario recordings to reproduce violations, and supports further analysis, including root cause investigation and ADS performance evaluation.

5 Evaluation and Results

To evaluate the effectiveness of SimADFuzz, we conducted experiments aimed at addressing the following research questions, covering various perspectives:

RQ1 To what extent can optimization strategies in scenario selection and mutation components of SimADFuzz improve its effectiveness in detecting violations?

RQ2 How effective is SimADFuzz in detecting violations in ADS compared to state-of-the-art fuzzers?

RQ3 Does the strategy used in SimADFuzz impact the diverse generation of scenarios?

5.1 Experimental Settings

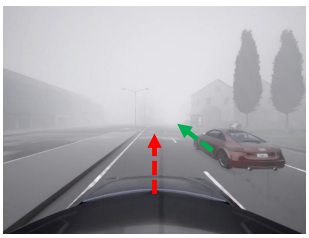
5.1.1 ADS Under Test. We select two ADS, *i.e.,* InterFuser [47] and LMDrive [46], for our experiments. InterFuser utilizes a Transformer-based architecture for interpretable sensor fusion, integrating sensor data to generate control commands for the ego vehicle. It has demonstrated its effectiveness in autonomous driving tasks by achieving high driving scores on the CARLA

Leaderboard. LMDrive [46] is a large language model based end-to-end ADS, which integrates multi-modal sensor data with natural language instructions, enhancing the ability of the ego vehicle to navigate complex scenarios. Additionally, both InterFuser and LMDrive are open-sourced and the weights of their models are publicly available. These features make them suitable choices for evaluating the performance of SimADFuzz.

5.1.2 Fuzzing Configurations. To evaluate the scenarios generated by SimADFuzz, we utilize the CARLA simulator, configured to run at a frame rate of 20 Hz. Each simulation lasts up to 10 minutes but may terminate earlier if the ego vehicle collides with other vehicles or pedestrians, or if it successfully reaches its destination.

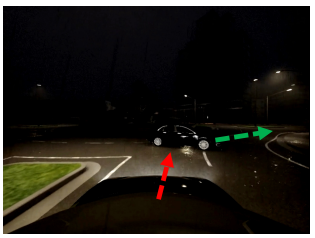
The maps used in our experiments are Town01 and Town03 provided by CARLA. Town01 is a small map with several bridges and intersections, while Town03 represents a large urban area resembling a downtown district, which includes roundabouts, underpasses, overpasses, and other complex road structures, offering diverse and challenging driving scenarios for testing.

Each testing scenario includes five pedestrians and three vehicles, one of which is configured as the ego vehicle deployed with InterFuser or LMDrive. Figure 7 illustrates six typical scenarios generated by SimADFuzz: two vehicles driving in the same or opposite directions, vehicles encountering each other at the roundabout, pedestrians crossing intersections under different lighting conditions, and multiple participants meeting at an intersection.



(a) Two vehicles driving in the same direction

(b) Two vehicles driving in the opposite direction



(c) Encountering vehicles at the roundabout

(d) Pedestrians crossing intersections

(e) Pedestrians crossing intersections in low-light conditions

(f) Multiple participants meeting at an intersection

Fig. 7. Scenarios Generated During the Fuzz Testing

The population size for fuzzing is set to 20, with crossover and mutation probabilities fixed at 0.5. To ensure a fair comparison across different fuzzing methods, the same seed scenarios are utilized for all experiments. These seed scenarios are generated by randomly selecting ego vehicle routes from the maps.

There are several hyperparameters in SimADFuzz, as shown in Table 5. The first three are related to the violation prediction model (VPM), including parameters for the Transformer encoder. The

remaining three are associated with mutation strategies, including two parameters mentioned in Algorithm 1 and the maximum distance between pedestrians and the ego vehicle when generating pedestrians.

Table 5. Hyperparameters of SimADFuzz

Category	Hyperparameter Description	Value
VPM-related	embedding dimension	128
	head number	3
	encoder layer number	3
Mutation-related	threshold for identifying stuck vehicles	10 meters
	time window size for identifying leaving vehicles	10 seconds
	maximum distance between pedestrian and the ego vehicle	20 meters

For the dataset generation, according to the empirical study conducted by Wang et al. [60], the improvement in quality of generated scenarios between 1,000 and 2,000 samples is not significant. Therefore, we generate 1,000 simple driving scenarios, each with a short route where the distance between the starting and ending points is 50 meters. We then collect feedback information to train both the violation prediction model and the SDC-Scissor model. For model training, the SDC-Scissor model utilizes a Logistic Regression classifier, which has been shown to perform effectively according to Birchler et al. [3]. The violation prediction model is trained using the Adam optimizer with a learning rate of 0.001 and binary cross-entropy as the loss function. To mitigate overfitting, we implement an early stopping strategy with a patience of 10 epochs. The training process for the violation prediction model requires approximately 5 minutes.

5.1.3 Baselines and Evaluation Metrics. We compare SimADFuzz against three different fuzzers:

AV-Fuzzer, which employs genetic algorithms to minimize the ego vehicle's safety potential. It generates offspring scenarios by altering the positions of NPCs. Note that AV-Fuzzer does not consider the number of unique violations in its fitness function by default. To ensure a fair comparison, we modify its fitness function to include the number of unique violations as an additional objective by a weighted sum ($0.5 \times$ original fitness + $0.5 \times$ number of unique violations).

DriveFuzz, which designs a fitness function based on hard acceleration and other behaviors to evaluate scenarios. It mutates the weather conditions, NPC positions, and pedestrian navigation types to generate diverse driving scenarios.

TM-Fuzzer, which dynamically manages traffic flow to increase interactions with the ego vehicle. It also incorporates clustering analysis to generate diverse test scenarios.

We use the number of unique violations (UVs) as the evaluation metric. UVs are defined as violations that occur at different times or locations. This metric is widely used in ADS fuzz testing [11, 28] and serves as a reliable indicator of a fuzzer's performance in detecting violations. In our evaluation, we define a temporal threshold of ±10 seconds and a spatial threshold of ±30 meters to determine unique violations.

5.2 Experimental Results

5.2.1 RQ1: Component Effectiveness. To validate the effectiveness of the violation prediction model, SDC-Scissor, and distance-guided mutation strategies, we conducted ablation experiments using SimADFuzz variants. All experiments were performed on the Town03 map with InterFuser as the test subject, where each variant was fuzzed for a total of 3 hours.

The results are shown in Figure 8, where each line represents a variant $X + Y$. Here, X indicates the components activated during scenario selection, which can be VS (using both the violation prediction model and SDC-Scissor, representing the complete fuzzer of SimADFuzz), V (using only the violation prediction model), S (using only SDC-Scissor), or R (randomly selecting scenarios). Y indicates the components activated during mutation, which can be D (using distance-guided mutation strategies) or R (randomly mutating scenarios).

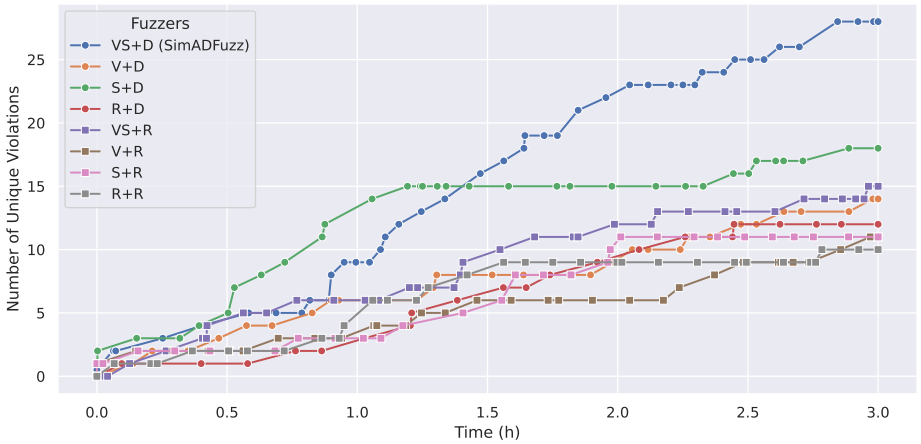


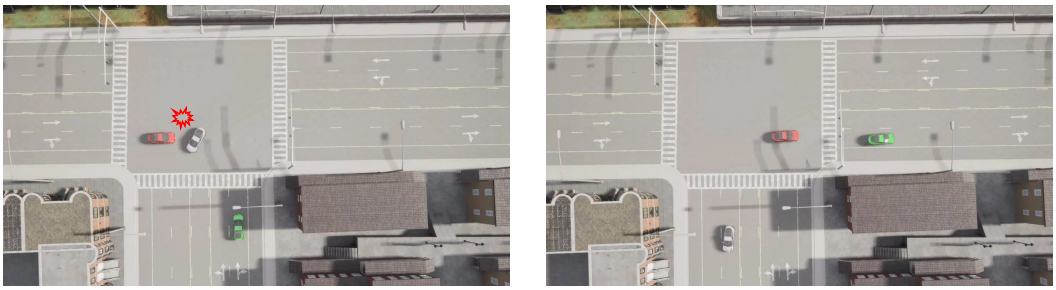
Fig. 8. The number of UVs detected by variants of SimADFuzz. Each variant is denoted as $X + Y$, where X represents the components activated during scenario selection (with V for the violation prediction model, S for SDC-Scissor, and R for random selection), and Y represents the components activated during mutation (with D for distance-guided mutation strategies and R for random mutation strategies).

As shown in Figure 8, the complete fuzzer $VS + D$ outperforms all other variants, discovering 28 UVs over a total of 3 hours of fuzzing, while the random baseline ($R + R$) only discovers 10 UVs. This indicates that the optimization components for both scenario selection and mutation greatly improve the ability to detect violations in ADS.

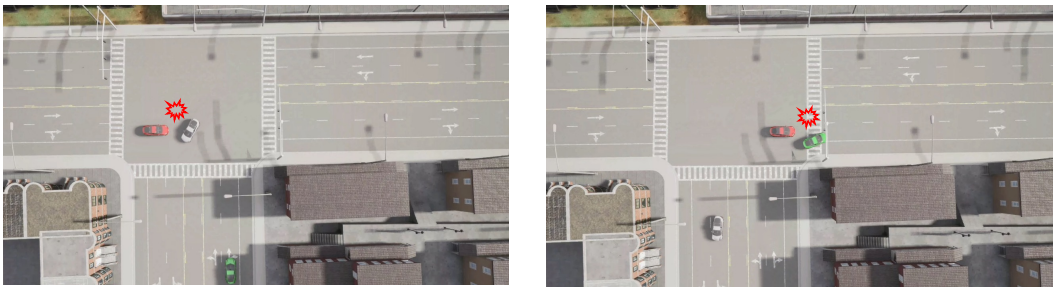
Focusing on scenario selection, $V+D$ and $S+D$ outperform $R+D$, indicating that both the violation prediction model and SDC-Scissor improve the effectiveness of selecting high-risk scenarios. The combination of these two models ($VS + D$) further refines the selection process by leveraging temporal features from the violation prediction model and static road features from SDC-Scissor.

To better demonstrate the effectiveness of the scenario selection strategy of SimADFuzz, we constructed two similar scenarios, S_1 and S_2, as described in the motivation examples (Section 3). Figure 9 shows the bird's eye view of these scenarios. It can be observed that scenario S_2 is riskier than S_1 because the green vehicle blocks the ego vehicle's path during right turns, potentially leading to rear-end collisions. We calculated the fitness scores of both scenarios using AV-Fuzzer, DriveFuzz, TM-Fuzzer and SimADFuzz. As shown in Table 6, AV-Fuzzer assigns a higher fitness score to S_1, while DriveFuzz and TM-Fuzzer consider both scenarios to be equally risky. Notably, DriveFuzz's objective values (*e.g.,* hard acceleration, hard braking, *etc.*) are identical in both scenarios. In contrast, SimADFuzz selects S_2 as the Pareto-optimal scenario through the NSGA-2 algorithm. The VPM predicts violation probabilities of 0.14 and 0.48 for S_1 and S_2, respectively, indicating that S_2 poses higher risk when spatial-temporal features are considered.

In terms of scenario mutation, variants with random mutation strategies ($VS+R$, $V+R$, $S+R$, and $R+R$) perform worse than their counterparts with distance-guided mutation strategies. Furthermore,



(a) Bird-view of Scenario S_1



(b) Bird-view of Scenario S_2

Fig. 9. Two similar scenarios corresponding to Figure 2a and Figure 2b. Each scenario is represented by a bird's eye view of the map, with the red vehicle is deployed with InterFuser as the ego vehicle. The differences between two scenarios are the start point of the green vehicle, where the start point of the green vehicle in S_2 is farther from the T-junction compared to S_1, causing the green vehicle blocks the ego vehicle during right turns, leading to a risky situation.

Table 6. The Fitness Score of Different Fuzzers in Two Similar Scenarios

Fuzzers	**S1**		**S2**	
	Object Values	**Fitness Score**	**Object Values**	**Fitness Score**
AV-Fuzzer	-	136.62	-	136.44
DriveFuzz	ha=14; hb=8; ht=0; os=0; us=0; 1/md=0.25	22.25	ha=14; hb=8; ht=0; os=0; us=0; 1/md=0.25	22.25
TM-Fuzzer	-md=-4.90; nova=0.47; distance=0	Pareto-optimal	-md=-4.73; nova=0.46; distance=0	Pareto-optimal
SimADFuzz	model=0.14; sdc=0.05; -md=-4.90; num_vio=0	Not Pareto-optimal	model=0.48; sdc=0.05; -md=-4.73; num_vio=0	Pareto-optimal

$VS + R$ shows stagnation in the number of unique violations detected during fuzzing. This suggests that random mutation strategies may generate low-quality scenarios with limited interaction between traffic participants. Table 7 supports this observation by showing that distance-guided mutation strategies increases both the number of NPC vehicles near the ego vehicle (*i.e.,* within 50 meters) and collision violations. Notably, while $VS + R$ only discovered one collision where the ego

vehicle hit roadside barriers without interacting with other participants, SimADFuzz uncovered more complex scenarios, including two collisions caused by NPC vehicles changing lanes and one collision involving a pedestrian crossing the road. These results demonstrate that the distance-based mutation strategies enhance vehicle interactions through increased proximity, which consequently affects the ADS's perception and decision-making, ultimately leading to more violations.

Table 7. The Number of NPC Vehicles Near by the Ego Vehicle and Collision Violations

Fuzzer variants	Fuzzing Time	Num of Vehicles	Num of Collision Violations
VS+R	1h	3	0
	2h	4	0
	3h	8	1
VS+D (SimADFuzz)	1h	7	2
	2h	11	3
	3h	15	3

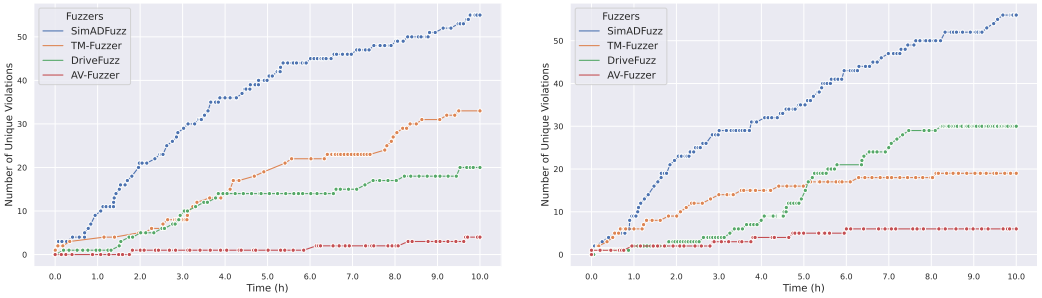
Answer to RQ1: The model-based fitness evaluation and distance-guided mutation strategies effectively enhance the performance of SimADFuzz, detecting 18 more unique violations compared to variants using random strategies over 3 hours fuzz testing.

5.2.2 RQ2: Performance Comparison with SOTA Fuzzers. We conducted experiments on InterFuser and LMDrive to compare the performance of SimADFuzz with other state-of-the-art fuzzers, including AV-Fuzzer, DriveFuzz, and TM-Fuzzer. Each experiments was executed for a total of 10 hours.

Figure 10 and Figure 11 shows the number of unique violations detected in InterFuser and LMDrive, respectively. The x-axis represents the fuzzing time in hours, while the y-axis indicates the number of unique violations detected. It can see that the performance gap between SimADFuzz and other fuzzers becomes increasingly significant as fuzzing progresses.

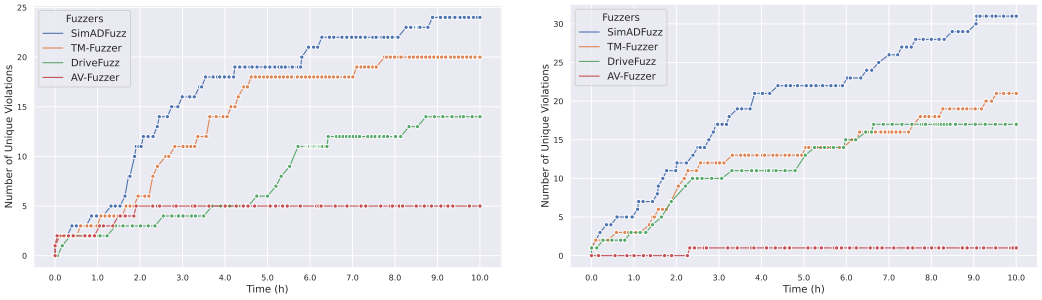
For InterFuser as the ADS under test, as shown in Figure 10, SimADFuzz detected a total of 111 UVs in two maps after 10 hours fuzzing, outperforming TM-Fuzzer (52 UVs), DriveFuzz (50 UVs), and AV-Fuzzer (10 UVs). Additionally, SimADFuzz identified its first collision violation within 4 minutes, compared to 7 minutes for TM-Fuzzer, demonstrates SimADFuzz's superior efficiency in detecting critical violations. Specifically, SimADFuzz detected 20 collisions, 54 lane invasions, and 37 stuck violations.

For another ADS under test, LMDrive, as shown in Figure 11, all four fuzzers discover fewer unique violations than InterFuser, since LMDrive is a more complex and advanced ADS with superior performance. According to the comparison results, SimADFuzz outperforms the other three baseline methods in both maps Town01 and Town03, with a total of 55 UVs detected, compared to 41 UVs for TM-Fuzzer, 31 UVs for DriveFuzz and 6 UVs for AV-Fuzzer. SimADFuzz triggers 19 lane invasion violations and 15 stuck violations; however, to our surprise, 21 collisions were detected in LMDrive, which is more than those detected in InterFuser. We manually checked the recordings and reran the scenarios generated by SimADFuzz, and found that LMDrive sometimes misjudged lane markings, leading to insufficient steering when turning, which results in the vehicle driving off the road and colliding with roadside barriers.



(a) Town01 (b) Town03

Fig. 10. The Number of UVs Detected by SimADFuzz and Baselines in Different Maps (InterFuser)



(a) Town01 (b) Town03

Fig. 11. The Number of UVs Detected by SimADFuzz and Baselines in Different Maps (LMDrive)

Notably, none of the fuzzers detected speeding or running red light violations during testing on InterFuser and LMDrive. We attribute this observation to the ADS's conservative configurations. Specifically, InterFuser is configured with a strict 5 m/s speed limit and a 0.3 confidence threshold for red traffic light detection. LMDrive acquires the real-time traffic light status directly from the simulator while maintaining smooth driving behavior through acceleration/deceleration rate constraints. These configurations make the ego vehicle behave as a cautious driver, avoiding speeding or moving when it is uncertain about the traffic light state.

To further analyze the results, we categorized the violations according to their types. Table 8 summarizes parts of violations discovered by SimADFuzz, briefly describing their scenarios and participants. Importantly, all scenarios were confirmed to be reproducible using SimADFuzz (detailed in Appendix A).

Regarding collision violations, we manually inspect each detected collision and categorize them into pedestrian, vehicle and roadside collisions. Below, we present five case studies (#1, #2, #3, #4 and #5 in Table 8), all of which can be reproduced using SimADFuzz.

Case Study#1: Pedestrian Collision by InterFuser. As shown in Figure 12, the collision occurs at night under low-light conditions. Although SimADFuzz ensures that vehicle headlights are activated when the sun altitude falls below 90 degrees, the driving conditions remain more challenging than in daylight. InterFuser successfully detects a moving object at timestamp t_1 (Figure 12(b)), but it fails to recognize the pedestrian at timestamps t_2 and t_3. From the control

Table 8. Violation Types and Scenario Descriptions

Violation Types	ID	Scenarios Description
Collision	#1	Ego vehicle failed to respond appropriately to a pedestrian jaywalking, resulting in a side collision.
	#2	Ego vehicle collides with another vehicle while changing lanes to exit a crossroads.
	#3	Ego vehicle collides with another vehicle while changing lanes at an intersection.
	#4	Ego vehicle collides with another vehicle due to insufficient steering while turning right.
	#5	Ego vehicle collides with roadside signboard due to misjudged lane markings.
Lane Invasion	#6	Ego vehicle illegally crosses the solid line while passing through an intersection.
	#7	Ego vehicle illegally crosses the solid line while turning right.
Stuck	#8	Ego vehicle failed to change lanes when the pedestrian in front remained stationary for too long.
	#9	Ego vehicle misjudged the state of the traffic lights, resulting in being stuck on a downhill ramp.

signals generated by InterFuser (highlighted in blue in the controller display at the bottom-right), we observe that the ego vehicle does not apply any braking at timestamps t_2 and t_3. Although the pedestrian's behavior contributed to the collision, InterFuser's failure to take appropriate actions (*i.e.,* braking) also makes it partially responsible for the accident.

Case Study#2, #3 and #4: Vehicle Collision by InterFuser. As shown in Figure 13, the ego vehicle rear-collides with a van when changing lanes to exit the roundabout. At the time, the van is stationary in the ego vehicle's path. From Figure 13(c), we can see that while InterFuser successfully detects an object to the right of the ego vehicle, it misjudges the size of the obstacle. This misjudgment results in insufficient left steering by the ego vehicle, ultimately causing the collision.

Figure 14 illustrates a collision caused by an improper lane change. At t_1, two vehicles are turning left at an intersection, with the front vehicle controlled by InterFuser. Then, at t_2, the ego vehicle merges into the side lane while a red car is driving alongside it on the left. Although InterFuser identified the red car, it failed to yield and persisted in making the lane change despite the situation. At t_3, a side collision occurred due to an incorrect estimation of the red vehicle's speed.

Figure 15 shows a more severe frontal collision. The ego vehicle, turning right at an intersection with insufficient steering, drives into the opposite lane. Unfortunately, a white truck is approaching from the opposite direction. Although the truck is braking and InterFuser immediately outputs a brake signal (as shown in Figure 15(c)), trying to avoid the collision by stopping in front of the truck. However, the ego vehicle's speed is too high to stop in time and ultimately, the two vehicles collide head-on.

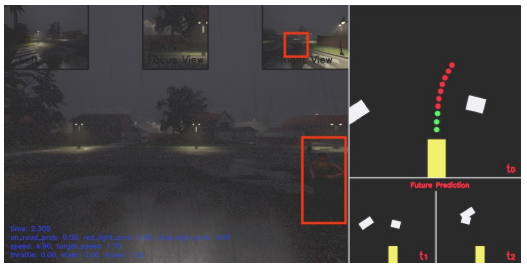
Case Study#5: Roadside Signboard Collision by LMDrive. Figure 16 illustrates a collision triggered by LMDrive under heavy fog conditions. During a left turn maneuver on a curved road, the system misjudged lane markings, causing the ego vehicle to deviate off-road as shown in Figure 16(c). The vehicle persistently maintained throttle input without braking or steering correction,[7] ultimately colliding with a roadside speed limit signboard at 14.65 km/h (Figure 16(d)).

Answer to RQ2: SimADFuzz reveals 150, 85 and 73 more unique violations than AV-Fuzzer, DriveFuzz and TM-Fuzzer respectively across two tested ADS, demonstrating superior effectiveness in generating high-quality scenarios for simulation-based testing.

5.2.3 RQ3: Diversity of Scenarios. The main goal of SimADFuzz is to generate more diverse and interactive driving scenarios. One of the metrics used to evaluate diversity is ego vehicle trajectory

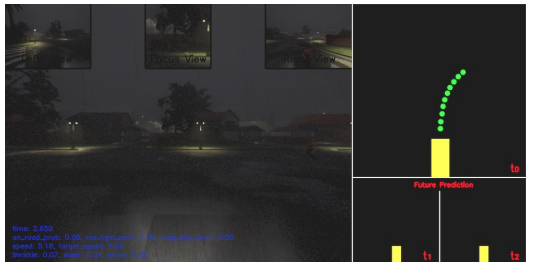
[7]The LMDrive controller's display interface contains a display mismatch: throttle and steering data are positionally swapped in the GUI.

(a) Bird-view of scenario at t_1



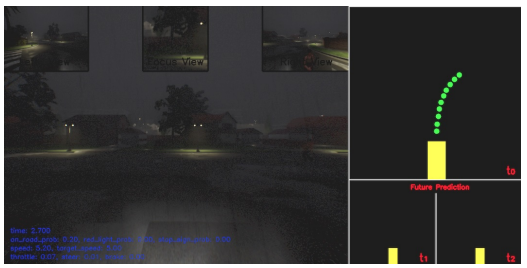
(b) InterFuser controller display at t_1

(c) Bird-view of scenario at t_2



(d) InterFuser controller display at t_2

(e) Bird-view of scenario at t_3

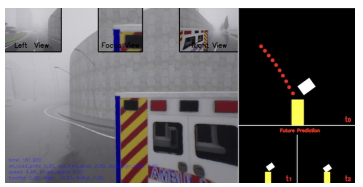


(f) InterFuser controller display at t_3

Fig. 12. Pedestrian Collision under Low-Light Conditions (#1)

(a) Bird-view of scenario

(b) Front-view of scenario



(c) InterFuser controller display

Fig. 13. Vehicle Collision during Lane Change at the Roundabout (#2)

coverage on the map [26]. We mark the waypoints on the Town03 map at 5-meter intervals and define trajectory coverage as the number of waypoints covered by the ego vehicle's route divided by the total number of waypoints on the map.

As shown in Figure 17, SimADFuzz (*i.e.,* VS+D) achieved 46.84% trajectory coverage, outperforming AV-Fuzzer (2.80%) which does not mutate the ego vehicle's route but only the NPCs, and

(a) Bird-view of scenario at t_1

(b) InterFuser controller display at t_1

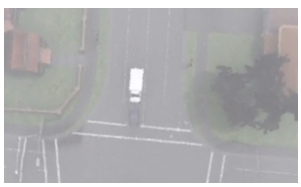
(c) Bird-view of scenario at t_2

(d) InterFuser controller display at t_2

(e) Bird-view of scenario at t_3

(f) InterFuser controller display at t_3

Fig. 14. Vehicle Collision during Lane Change at the Intersection (#3)



(a) Bird-view of scenario

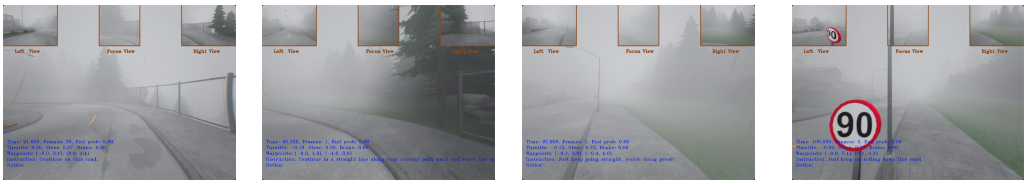
(b) Front-view of scenario

(c) InterFuser controller display

Fig. 15. Vehicle Collision during Turn Right (#4)

DriveFuzz (12.04%) which mutates the ego vehicle's route only after the cycle and mutation process are completed. TM-Fuzzer, which aims to increase interactions by dynamically controlling traffic flow, achieved a trajectory coverage of 17.88%.

Furthermore, the two variants of SimADFuzz, $R + R$ and $VS + R$, achieve 23.18% and 33.10% of trajectory coverage, respectively. Although random-based scenario mutation strategies have a



(a) Before Turning (b) Turning Left (c) Off Road (d) Collision with Signboard

Fig. 16. Roadside Signboard Collision during Turn Left (#5)

higher probability of mutating the ego vehicles to discover more map waypoints than SimADFuzz, which focuses on mutating NPC vehicles, $R + R$ and $VS + R$ often assign longer driving tasks for the ego vehicle, reducing simulation efficiency. For example, we counted the number of scenarios executed by SimADFuzz and $VS + R$ during the 3-hour fuzzing process. The results show that SimADFuzz executed 48 scenarios, while $VS + R$ executed 37 scenarios. This difference explains why the map coverage of $R + R$ and $VS + R$ is lower than that of SimADFuzz.

Recall that SimADFuzz detects more violations than other fuzzers, which indicates that SimADFuzz not only increases the diversity of the ego vehicle's trajectory across the map but also improves the quality of the scenarios for detecting risky behaviors.

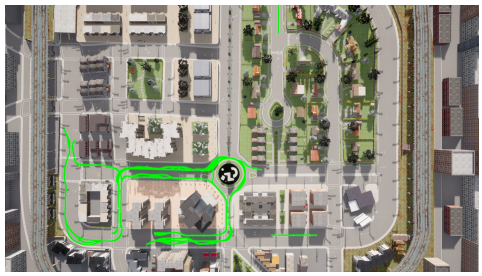
Answer to RQ3: The strategies used in SimADFuzz enhance the coverage of the ego vehicle's trajectory on the map, thereby improving the diversity of the generated scenarios.

5.3 Threats to Validity

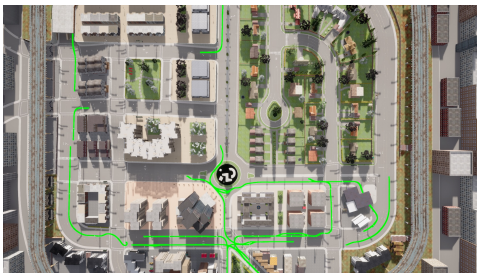
5.3.1 Internal Validity. One potential threat to internal validity is the implementation of SimADFuzz. We developed SimADFuzz based on the DEAP and CARLA simulator and extended it by (1) enhancing the simulation information collector through analysis of actions determined by the ADS, (2) integrating a model-based fitness evaluation using a deep neural network model and adapting the fitness score to the NSGA-2 algorithm provided by DEAP, and (3) customizing the scenario mutation procedure by introducing a distance-guided mutation strategy. Although the implementation of SimADFuzz has undergone peer review, there may still be issues that could affect the experimental results. To mitigate this risk, we have made the source code of SimADFuzz publicly available (details in Appendix A), allowing the community to reproduce the results and validate the implementation.

5.3.2 External Validity. A potential threat to external validity is that we evaluate the effectiveness of SimADFuzz primarily based on InterFuser and LMDrive using two simulation maps, which may limit the generalizability of the results to other ADSs or driving environments. Specifically, industrial-grade ADSs (*e.g.,* Apollo and Autoware) generally have more complex architectures than research-prototype ADSs, and certain driving scenarios (*e.g.,* parking, U-turns, etc.) are not covered by the two maps used in our experiments. Therefore, the effectiveness of SimADFuzz on these industrial-grade ADSs and in such specific scenarios requires further investigation in future work. Nevertheless, it is important to note that InterFuser has achieved top-performing results on the CARLA Leaderboard, and LMDrive integrates complex large language models, representing novel and advanced ADSs. Additionally, the two selected maps adequately cover typical driving environments in both rural and urban areas, and both the test subjects and simulation maps have been widely employed in previous works [29, 34, 60, 68], which mitigates threats to external validity.

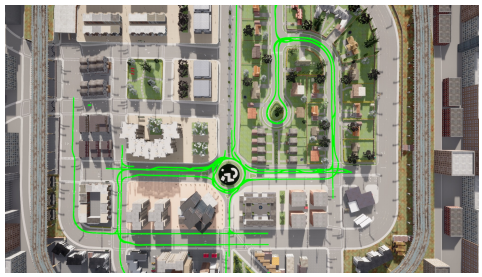
(a) AV-Fuzzer (2.80%)



(b) DriveFuzz (12.04%)

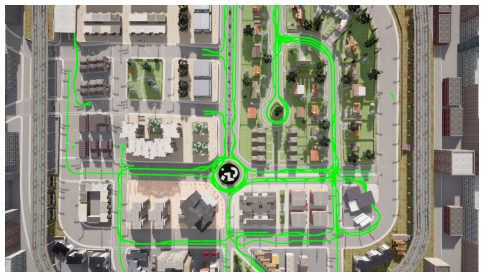


(c) TM-Fuzzer (17.88%)



(d) SimADFuzz-R+R (23.18%)

(e) SimADFuzz-VS+R (33.10%)



(f) SimADFuzz-VS+D (46.84%)

Fig. 17. Trajectory Coverage of Fuzzers in Town03

5.3.3 Construct Validity. The primary threat to construct validity lies in the metrics used to evaluate the effectiveness of SimADFuzz. In our experiment, we focus solely on the number of unique violations detected by each fuzzer. The violation oracle used in SimADFuzz, its variants, and other fuzzers are identical. Therefore, comparing the number of unique violations provides a fair assessment of the performance of SimADFuzz. Furthermore, the number of unique violations is a widely accepted metric in the evaluation of fuzz testing for ADS [11, 28], and it effectively reflects the capability of SimADFuzz in detecting violations within ADS.

6 Related Work

SimADFuzz leverages fuzzing techniques for simulation-based testing, and proposes novel scenario selection and mutation strategies to generate offspring scenarios. This section reviews the related work on simulation-based testing for ADS and optimization techniques for fuzz testing.

6.1 Simulation-based Testing for ADS

Simulation-based testing presents a viable and efficient alternative, enabling the exploration of a broad spectrum of scenarios and environments in a controlled and safe setting [42]. However, as the complexity of simulated environments increases, the number of configurable and variable elements also grows, leading to a vast space of possible scenarios. Therefore, simulation-based testing methods for ADS focus on how to generate critical scenarios.

Wang et al. [58] presented AdvSim, a framework that generates safety-critical scenarios by adversarially altering the trajectories of actors within traffic scenarios to test the LiDAR-based ADS. Gambi et al. [18] presented ASFault, which creates road networks to simulate driving scenarios and employs a genetic algorithm to generate tests aimed at exposing unsafe behaviors in lane-keeping systems.

Moreover, Sun et al. [52] and Zhang et al. [67] expanded the oracle of violations beyond collisions to traffic laws. They presented LawBreaker and GFlowNet, respectively, which assess the "distance" between the behavior of ego vehicles and traffic laws using signal temporal logic [38], thereby generating scenarios that more closely resemble violations. Huai et al. [28] presented Doppel, which replaces non-intelligent agents in the simulation with the ego vehicle, ensuring that all violations are triggered by the ego vehicle and reducing false positives. Lu et al. [36] proposed DeepCollision based on the Deep Q-Learning (DQN) algorithm [40], which calculates the safety distance and current distance to evaluate the collision probability, and designs reward functions to make the DQN output the optimal action, thereby increasing the possibility of detecting collision behaviors. Tian et al. proposed MOSAT [53] and CRISCO [54]. MOSAT represents vehicle driving behavior as a gene sequence composed of atomic driving maneuvers. It considers fitness metrics such as the estimation time to collision (ETTC) and leverage NSGA-2 to select scenarios, then the maneuver sequence are mutated to generate offspring driving scenarios. CRISCO extracts influential behavior patterns from historical traffic accidents, and assigns participants to move along specified trajectories during scenario generation. Additionally, CRISCO leverages ETTC to evaluate the criticality of driving scenarios, selecting more critical scenarios to challenge the ADS.

Haq et al. [24] focus on DNN-enabled systems and propose an online testing method that utilizes MOSA/FITEST as the multi-objective search algorithm to guide the generation of test cases. In contrast, SimADFuzz employs the NSGA-2 algorithm, which is orthogonal to MOSA/FITEST. While both aim to optimize multiple objectives, they do so through different methodologies and strategies, offering unique advantages in different contexts, such as exploration vs. exploitation trade-offs.

AV-Fuzzer [31], AutoFuzz [68], DriveFuzz [30], TM-Fuzzer [34], scenoRITA [27] and Scenario-Fuzz [60] are representative methods that utilize fuzz testing techniques to generate scenarios for autonomous driving simulations. These fuzzers adapt their scenario generation strategies based on simulation feedback. For instance, AV-Fuzzer retains scenarios with higher potential safety risks when interacting with other actors. AutoFuzz evaluates objectives such as the ego vehicle's speed during collisions and the minimum distance to other actors. Similarly, DriveFuzz incorporates metrics such as hard braking, steering, and minimum distances as driving quality scores, selecting the scenarios with the highest fitness to generate offspring. TM-Fuzzer utilizes real-time traffic management and diversity analysis to serach and generate critical and unique scenarios. scenoRITA optimize mutation strategies force on obstacles states (*e.g.,* vehicles, pedestrians and bikes) and proposed a technique to eliminate duplicate scenarios to enhance testing efficiency. The most similar work with SimADFuzz is ScenarioFuzz, which builds a graph neural network to predict and filter out high-risk scenarios, while we leverage Transformer model to embed the scenarios by considering the temporal features of the ego vehicle and NPC vehicles. Note that SimADFuzz

selects scenarios based on multi-objective search algorithm NSGA-2, so it is convenient to adapt other fitness (such as confidence level proposed by ScenarioFuzz) to further refine our method.

6.2 Optimization Techniques for Fuzz Testing

Fuzz testing is a dynamic software analysis and testing technique that employs random inputs as test cases, which are then executed within the programs under test (PUT) [33]. The quality of test cases is pivotal to the efficacy of fuzz testing; high-quality test cases facilitate the exploration of more diverse execution paths within the program, while low-quality test cases may lead to inefficient resource allocation and reduced testing effectiveness. Thus, recent work focused on optimizing seed selection and mutation strategies to generate high-quality test cases.

For seed selection, Memlock [61] prioritizes seeds based on coverage and memory consumption to uncover vulnerabilities related to memory usage in PUT. Truzz [66] favors seeds that generate new edge coverages, thereby improving the code coverage. K-Scheduler [50] assigns different probabilities for selection based on the centrality of each seed. Cerebro [32] calculates a comprehensive score for seeds based on the complexity of unexplored code near the execution path. AFLSmart [43] computes the effectiveness ratio of seeds, with higher effectiveness seeds being allocated more resources, thus increasing the likelihood of generating effective offspring test cases.

For seed mutation, several works leverage machine learning models to optimize mutation strategies. Neuzz [49] employs a neural network model to smoothen complex PUT, establishing a relationship between test case byte sequences and branch coverage, identifying key byte positions through gradients for targeted mutation. MTFuzz [48] uses a Multi-Task neural network to learn a compact embedding of the input space for multiple related tasks, guiding the mutation process by focusing on high-gradient areas of the embedding. Wu et al. [62] proposed PreFuzz, which enhances gradient guidance through a resource-efficient edge selection mechanism and a probabilistic byte selection mechanism to improve mutation effectiveness.

Existing seed selection and mutation optimization strategies cannot be directly applied to fuzz testing for ADS, because the scenarios are well-defined structured data compared to byte sequences. Inspired by SmarTest [51], which leverages language models to prioritize and generate transaction sequences (*i.e.,* test cases in the context of smart contracts), SimADFuzz employs a model-based fitness score evaluation method to optimize scenario selection. Furthermore, by measuring the distance between the ego vehicle and other traffic participants, the mutation strategy is refined to increase the probabilities of interactions between vehicles, leading to more effective scenario generation.

7 Conclusion

In this paper, we have proposed a novel fuzz testing method SimADFuzz, which addressed the limitations of existing methods and generated high-quality scenarios for autonomous driving systems. SimADFuzz combines a model-based fitness evaluation approach with distance-guided mutation strategies to improve the ability of fuzz testing in detecting violations. We have conducted extensive experiments to evaluate the effectiveness of SimADFuzz. The results show that compared to the state-of-art approaches AV-Fuzzer, DriveFuzz and TM-Fuzzer, SimADFuzz can detect 150, 85 and 73 more violations and demonstrate its ability to identify potential issues in the ADS.

In future work, we aim to further enhance scenario selection and mutation strategies. For example, we plan to incorporate factors such as traffic signals and weather conditions into the neural network model to improve the accuracy of fitness value evaluation. In addition, we intend to explore the use of sequence models such as Informer [70], which has demonstrated superior performance over Transformers in extracting temporal features from trajectory prediction [8], to enhance the effectiveness of the scenario violation prediction model. Future work also includes investigating

replacing NSGA-2 with advanced multi-objective algorithms such as MOPSO [12] and NSGA-3 [14] to analyze differences in performance. Furthermore, we shall consider evaluating SimADFuzz on industrial-grade ADSs, such as Apollo and Autoware, which offer more complex and realistic system architectures. Moreover, experiments will be extended to larger and more diverse simulation maps, enabling more comprehensive assessment of SimADFuzz's effectiveness under varying and challenging driving scenarios.

Acknowledgments

The authors are grateful for the valuable feedback from reviewers. This work was partially supported by the National Natural Science Foundation of China (NSFC, No. 62372232), and the Collaborative Innovation Center of Novel Software Technology and Industrialization. T. Chen is partially supported by an oversea grant from the State Key Laboratory of Novel Software Technology, Nanjing University (KFKT2023A04, KFKT2025A05).

A Data Availability

The source code of SimADFuzz, along with the reproducible violation scenarios listed in Table 8, is available at https://github.com/yagol2020/SimADFuzz.

References

[1] Bushra Alhijawi and Arafat Awajan. 2023. Genetic algorithms: Theory, genetic operators, solutions, and applications. *Evolutionary Intelligence* (2023), 1–12.

[2] Baidu Apollo. 2023. Dreamview. https://developer.apollo.auto/platform/simulation.html.

[3] Christian Birchler, Nicolas Ganz, Sajad Khatiri, Alessio Gambi, and Sebastiano Panichella. 2022. Cost-effective simulation-based test selection in self-driving cars software with SDC-Scissor. In *2022 IEEE international conference on software analysis, evolution and reengineering (SANER)*. IEEE, 164–168.

[4] Christian Birchler, Sajad Khatiri, Bill Bosshard, Alessio Gambi, and Sebastiano Panichella. 2023. Machine learning-based test selection for simulation-based testing of self-driving cars software. *Empirical Software Engineering* 28, 3 (2023), 71. https://doi.org/10.1007/s10664-023-10286-y

[5] Marcel Böhme, Van-Thuan Pham, Manh-Dung Nguyen, and Abhik Roychoudhury. 2017. Directed Greybox Fuzzing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 2329–2344. https://doi.org/10.1145/3133956.3134020

[6] Ezequiel Castellano, Ahmet Cetinkaya, and Paolo Arcaini. 2021. Analysis of Road Representations in Search-Based Testing of Autonomous Driving Systems. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*. 167–178. https://doi.org/10.1109/QRS54544.2021.00028

[7] Ching-Yao Chan. 2017. Advancements, prospects, and impacts of automated driving systems. *International journal of transportation science and technology* 6, 3 (2017), 208–216.

[8] Chongpu Chen, Xinbo Chen, Chong Guo, and Peng Hang. 2023. Trajectory Prediction for Autonomous Driving Based on Structural Informer Method. *IEEE Transactions on Automation Science and Engineering* (2023), 1–12. https://doi.org/10.1109/TASE.2023.3342978

[9] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. 2015. DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

[10] Yuntianyi Chen, Yuqi Huai, Shilong Li, Changnam Hong, and Joshua Garcia. 2024. Misconfiguration Software Testing for Failure Emergence in Autonomous Driving Systems. *Proc. ACM Softw. Eng.* 1, FSE, Article 85 (July 2024), 24 pages. https://doi.org/10.1145/3660792

[11] Mingfei Cheng, Yuan Zhou, and Xiaofei Xie. 2023. Behavexplor: Behavior diversity guided testing for autonomous driving systems. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 488–500.

[12] C.A. Coello Coello and M.S. Lechuga. 2002. MOPSO: a proposal for multiple objective particle swarm optimization. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, Vol. 2. 1051–1056 vol.2. https://doi.org/10.1109/CEC.2002.1004388

[13] Jiarun Dai, Bufan Gao, Mingyuan Luo, Zongan Huang, Zhongrui Li, Yuan Zhang, and Min Yang. 2024. SCTrans: Constructing a Large Public Scenario Dataset for Simulation Testing of Autonomous Driving Systems. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–13.

[14] Kalyanmoy Deb and Himanshu Jain. 2014. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints. *IEEE Transactions on Evolutionary Computation* 18, 4 (2014), 577–601. https://doi.org/10.1109/TEVC.2013.2281535

[15] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An open urban driving simulator. In *Conference on robot learning*. PMLR, 1–16.

[16] Vinicius H. S. Durelli, Rafael S. Durelli, Simone S. Borges, Andre T. Endo, Marcelo M. Eler, Diego R. C. Dias, and Marcelo P. Guimarães. 2019. Machine Learning Applied to Software Testing: A Systematic Mapping Study. *IEEE Transactions on Reliability* 68, 3 (2019), 1189–1212. https://doi.org/10.1109/TR.2019.2892517

[17] Alessio Gambi, Tri Huynh, and Gordon Fraser. 2019. Generating effective test cases for self-driving cars from police reports. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 257–267. https://doi.org/10.1145/3338906.3338942

[18] Alessio Gambi, Marc Müller, and Gordon Fraser. 2019. Asfault: Testing self-driving car software using search-based procedural content generation. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 27–30.

[19] Hongbo Gao, Hang Su, Yingfeng Cai, Renfei Wu, Zhengyuan Hao, Yongneng Xu, Wei Wu, Jianqing Wang, Zhijun Li, and Zhen Kan. 2021. Trajectory prediction of cyclist based on dynamic Bayesian network and long short-term memory model at unsignalized intersections. *Science China Information Sciences* 64, 7 (2021), 172207.

[20] Zhenhai Gao, Mingxi Bao, Fei Gao, and Minghong Tang. 2023. Probabilistic multi-modal expected trajectory prediction based on LSTM for autonomous driving. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering* (2023), 09544070231167906.

[21] Joshua Garcia, Yang Feng, Junjie Shen, Sumaya Almanee, Yuan Xia, and Qi Alfred Chen. 2020. A comprehensive study of autonomous vehicle bugs. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 385–396. https://doi.org/10.1145/3377811.3380397

[22] Maosi Geng, Junyi Li, Yingji Xia, and Xiqun Michael Chen. 2023. A physics-informed Transformer model for vehicle trajectory prediction on highways. *Transportation research part C: emerging technologies* 154 (2023), 104272.

[23] Francesco Giuliari, Irtiza Hasan, Marco Cristani, and Fabio Galasso. 2021. Transformer Networks for Trajectory Forecasting. In *2020 25th International Conference on Pattern Recognition (ICPR)*. 10335–10342. https://doi.org/10.1109/ICPR48806.2021.9412190

[24] Fitash Ul Haq, Donghwan Shin, and Lionel C. Briand. 2022. Efficient Online Testing for DNN-Enabled Systems using Surrogate-Assisted and Many-Objective Optimization.. In *ICSE*. ACM, 811–822. http://dblp.uni-trier.de/db/conf/icse/icse2022.html#Haq0B22

[25] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[26] Zhisheng Hu, Shengjian Guo, Zhenyu Zhong, and Kang Li. 2021. Coverage-based scene fuzzing for virtual autonomous driving testing. *arXiv preprint arXiv:2106.00873* (2021).

[27] Yuqi Huai, Sumaya Almanee, Yuntianyi Chen, Xiafa Wu, Qi Alfred Chen, and Joshua Garcia. 2023. scenoRITA: Generating Diverse, Fully Mutable, Test Scenarios for Autonomous Vehicle Planning. *IEEE Transactions on Software Engineering* 49, 10 (2023), 4656–4676. https://doi.org/10.1109/TSE.2023.3309610

[28] 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 *Proceedings of the 45th International Conference on Software Engineering* (Melbourne, Victoria, Australia) *(ICSE '23)*. IEEE Press, 2591–2603. https://doi.org/10.1109/ICSE48619.2023.00216

[29] Xinyu Ji, Lei Xue, Zhijian He, and Xiapu Luo. 2025. Autonomous Driving System Testing via Diversity-Oriented Driving Scenario Exploration. *ACM Transactions on Software Engineering and Methodology* (2025).

[30] Seulbae Kim, Major Liu, Junghwan "John" Rhee, Yuseok Jeon, Yonghwi Kwon, and Chung Hwan Kim. 2022. DriveFuzz: Discovering Autonomous Driving Bugs through Driving Quality-Guided Fuzzing. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 1753–1767. https://doi.org/10.1145/3548606.3560558

[31] Guanpeng Li, Yiran Li, Saurabh Jha, Timothy Tsai, Michael Sullivan, Siva Kumar Sastry Hari, Zbigniew Kalbarczyk, and Ravishankar Iyer. 2020. AV-FUZZER: Finding Safety Violations in Autonomous Driving Systems. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. 25–36. https://doi.org/10.1109/ISSRE5003.2020.00012

[32] Yuekang Li, Yinxing Xue, Hongxu Chen, Xiuheng Wu, Cen Zhang, Xiaofei Xie, Haijun Wang, and Yang Liu. 2019. Cerebro: context-aware adaptive fuzzing for effective vulnerability detection. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 533–544.

[33] Hongliang Liang, Xiaoxiao Pei, Xiaodong Jia, Wuwei Shen, and Jian Zhang. 2018. Fuzzing: State of the art. *IEEE Transactions on Reliability* 67, 3 (2018), 1199–1218.

[34] Shenghao Lin, Fansong Chen, Laile Xi, Gaosheng Wang, Rongrong Xi, Yuyan Sun, and Hongsong Zhu. 2024. TM-fuzzer: fuzzing autonomous driving systems through traffic management. *Automated Software Engineering* 31, 2 (2024), 61.

[35] Guannan Lou, Yao Deng, Xi Zheng, Mengshi Zhang, and Tianyi Zhang. 2022. Testing of autonomous driving systems: where are we and where should we go?. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Singapore) *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 31–43. https://doi.org/10.1145/3540250.3549111

[36] Chengjie Lu, Yize Shi, Huihui Zhang, Man Zhang, Tiexin Wang, Tao Yue, and Shaukat Ali. 2023. Learning Configurations of Operating Environment of Autonomous Vehicles to Maximize their Collisions. *IEEE Transactions on Software Engineering* 49, 1 (2023), 384–402. https://doi.org/10.1109/TSE.2022.3150788

[37] Xiaolei Ma, Zhimin Tao, Yinhai Wang, Haiyang Yu, and Yunpeng Wang. 2015. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. *Transportation Research Part C: Emerging Technologies* 54 (2015), 187–197.

[38] Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*. Springer, 152–166.

[39] Valentin J.M. Manès, HyungSeok Han, Choongwoo Han, Sang Kil Cha, Manuel Egele, Edward J. Schwartz, and Maverick Woo. 2021. The Art, Science, and Engineering of Fuzzing: A Survey. *IEEE Transactions on Software Engineering* 47, 11 (2021), 2312–2331. https://doi.org/10.1109/TSE.2019.2946563

[40] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[41] Tobias Moers, Lennart Vater, Robert Krajewski, Julian Bock, Adrian Zlocki, and Lutz Eckstein. 2022. The exiD Dataset: A Real-World Trajectory Dataset of Highly Interactive Highway Scenarios in Germany. In *2022 IEEE Intelligent Vehicles Symposium (IV)*. 958–964. https://doi.org/10.1109/IV51971.2022.9827305

[42] Demin Nalic, Tomislav Mihalj, Maximilian Bäumler, Matthias Lehmann, Arno Eichberger, and Stefan Bernsteiner. 2020. Scenario based testing of automated driving systems: A literature survey. In *FISITA web Congress*, Vol. 10. 1.

[43] Van-Thuan Pham, Marcel Böhme, Andrew E Santosa, Alexandru Răzvan Căciulescu, and Abhik Roychoudhury. 2019. Smart greybox fuzzing. *IEEE Transactions on Software Engineering* 47, 9 (2019), 1980–1997.

[44] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, et al. 2020. Lgsvl simulator: A high fidelity simulator for autonomous driving. In *2020 IEEE 23rd International conference on intelligent transportation systems (ITSC)*. IEEE, 1–6.

[45] Barbara Schütt, Markus Steimle, Birte Kramer, Danny Behnecke, and Eric Sax. 2022. A Taxonomy for Quality in Simulation-Based Development and Testing of Automated Driving Systems. *IEEE Access* 10 (2022), 18631–18644. https://doi.org/10.1109/ACCESS.2022.3149542

[46] Hao Shao, Yuxuan Hu, Letian Wang, Guanglu Song, Steven L Waslander, Yu Liu, and Hongsheng Li. 2024. Lmdrive: Closed-loop end-to-end driving with large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 15120–15130.

[47] Hao Shao, Letian Wang, Ruobing Chen, Hongsheng Li, and Yu Liu. 2023. Safety-enhanced autonomous driving using interpretable sensor fusion transformer. In *Conference on Robot Learning*. PMLR, 726–737.

[48] Dongdong She, Rahul Krishna, Lu Yan, Suman Jana, and Baishakhi Ray. 2020. MTFuzz: fuzzing with a multi-task neural network. In *Proceedings of the 28th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 737–749.

[49] Dongdong She, Kexin Pei, Dave Epstein, Junfeng Yang, Baishakhi Ray, and Suman Jana. 2019. Neuzz: Efficient fuzzing with neural program smoothing. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 803–817.

[50] Dongdong She, Abhishek Shah, and Suman Jana. 2022. Effective seed scheduling for fuzzing with graph centrality analysis. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2194–2211.

[51] Sunbeom So, Seongjoon Hong, and Hakjoo Oh. 2021. SmarTest: Effectively Hunting Vulnerable Transaction Sequences in Smart Contracts through Language Model-Guided Symbolic Execution. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 1361–1378. https://www.usenix.org/conference/usenixsecurity21/presentation/so

[52] Yang Sun, Christopher M Poskitt, Jun Sun, Yuqi Chen, and Zijiang Yang. 2022. LawBreaker: An approach for specifying traffic laws and fuzzing autonomous vehicles. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.

[53] Haoxiang Tian, Yan Jiang, Guoquan Wu, Jiren Yan, Jun Wei, Wei Chen, Shuo Li, and Dan Ye. 2022. MOSAT: finding safety violations of autonomous driving systems using multi-objective genetic algorithm. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Singapore, Singapore) *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 94–106. https:

//doi.org/10.1145/3540250.3549100

[54] Haoxiang Tian, Guoquan Wu, Jiren Yan, Yan Jiang, Jun Wei, Wei Chen, Shuo Li, and Dan Ye. 2023. Generating Critical Test Scenarios for Autonomous Driving Systems via Influential Behavior Patterns. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering* (Rochester, MI, USA) *(ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 46, 12 pages. https://doi.org/10.1145/3551349.3560430

[55] tier4. 2023. AWSIM. https://github.com/tier4/AWSIM.

[56] Simon Ulbrich, Till Menzel, Andreas Reschka, Fabian Schuldt, and Markus Maurer. 2015. Defining and Substantiating the Terms Scene, Situation, and Scenario for Automated Driving. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. 982–988. https://doi.org/10.1109/ITSC.2015.164

[57] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).

[58] Jingkang Wang, Ava Pun, James Tu, Sivabalan Manivasagam, Abbas Sadat, Sergio Casas, Mengye Ren, and Raquel Urtasun. 2021. AdvSim: Generating Safety-Critical Scenarios for Self-Driving Vehicles. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 9904–9913. https://doi.org/10.1109/CVPR46437.2021.00978

[59] Sen Wang, Zhuheng Sheng, Jingwei Xu, Taolue Chen, Junjun Zhu, Shuhui Zhang, Yuan Yao, and Xiaoxing Ma. 2022. ADEPT: A Testing Platform for Simulated Autonomous Driving. In *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*. ACM, 150:1–150:4. https://doi.org/10.1145/3551349.3559528

[60] Tong Wang, Taotao Gu, Huan Deng, Hu Li, Xiaohui Kuang, and Gang Zhao. 2024. Dance of the ADS: Orchestrating Failures through Historically-Informed Scenario Fuzzing. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 1086–1098.

[61] Cheng Wen, Haijun Wang, Yuekang Li, Shengchao Qin, Yang Liu, Zhiwu Xu, Hongxu Chen, Xiaofei Xie, Geguang Pu, and Ting Liu. 2020. Memlock: Memory usage guided fuzzing. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 765–777.

[62] Mingyuan Wu, Ling Jiang, Jiahong Xiang, Yuqun Zhang, Guowei Yang, Huixin Ma, Sen Nie, Shi Wu, Heming Cui, and Lingming Zhang. 2022. Evaluating and improving neural program-smoothing-based fuzzing. In *Proceedings of the 44th International Conference on Software Engineering*. 847–858.

[63] Yufei Xu, Yu Wang, and Srinivas Peeta. 2023. Leveraging transformer model to predict vehicle trajectories in congested urban traffic. *Transportation research record* 2677, 2 (2023), 898–909.

[64] Michał Zalewski. 2013. American Fuzzy Lop. http://lcamtuf.coredump.cx/afl/.

[65] Chi Zhang, Yuehu Liu, Danchen Zhao, and Yuanqi Su. 2014. RoadView: A traffic scene simulator for autonomous vehicle simulation testing. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. 1160–1165. https://doi.org/10.1109/ITSC.2014.6957844

[66] Kunpeng Zhang, Xi Xiao, Xiaogang Zhu, Ruoxi Sun, Minhui Xue, and Sheng Wen. 2022. Path transitions tell more: Optimizing fuzzing schedules via runtime program states. In *Proceedings of the 44th International Conference on Software Engineering*. 1658–1668.

[67] Xiaodong Zhang, Wei Zhao, Yang Sun, Jun Sun, Yulong Shen, Xuewen Dong, and Zijiang Yang. 2023. Testing automated driving systems by breaking many laws efficiently. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 942–953.

[68] Ziyuan Zhong, Gail Kaiser, and Baishakhi Ray. 2022. Neural network guided evolutionary fuzzing for finding traffic violations of autonomous vehicles. *IEEE Transactions on Software Engineering* (2022).

[69] Husheng Zhou, Wei Li, Zelun Kong, Junfeng Guo, Yuqun Zhang, Bei Yu, Lingming Zhang, and Cong Liu. 2020. DeepBillboard: systematic physical-world testing of autonomous driving systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 347–358. https://doi.org/10.1145/3377811.3380422

[70] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 11106–11115.

[71] Eckart Zitzler and Lothar Thiele. 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE transactions on Evolutionary Computation* 3, 4 (1999), 257–271.