UNIVERSITY OF CALIFORNIA,
IRVINE

Transferring A Testing Technique Among Autonomous Driving Systems

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Software Engineering

by

Shilong Li

2024

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I am deeply grateful to Professor Joshua Garcia, Professor James Jones, Professor Alfred Chen, and all the professors and friends who provided support during my master's studies. Your guidance and encouragement have been invaluable throughout this journey.

I also extend special thanks to my colleagues at UC Irvine for their collaboration and insightful feedback, which were essential in refining the concepts and methodologies presented in this work.

In addition, I appreciate the developers of Autoware for their prompt and helpful responses to our inquiries.

ABSTRACT OF THE THESIS

Transferring A Testing Technique Among Autonomous Driving Systems

By

Shilong Li

Master of Science in Software Engineering

University of California, Irvine, 2024

Assistant Professor Joshua Garcia, Chair

Autonomous vehicles (AVs) are increasingly prevalent in our daily routines, with examples such as robotaxis and robot deliveries becoming more commonplace. This trend emphasizes the importance of thorough testing procedures to guarantee their safety and dependability. Autonomous driving software (ADS) enables AVs to navigate using complex algorithms and machine learning techniques. While field operational tests are regularly employed to assess ADS functionality, they are hindered by cost and geographic limitations. Virtual testing emerges as a safer and more controlled alternative. Past studies have primarily focused on validating their approaches on Baidu Apollo, with limited exploration of another open-source ADS, Autoware. This raises questions about the generalizability of existing methodologies across different autonomous driving systems. To address this gap, this study aims to transfer scenoRITA to generate safety-critical and motion sickness-inducing test scenarios for Autoware. Our empirical results reveal that scenoRITA identifies 63 unique safety and comfort violations in Autoware.

Chapter 1

Introduction

Autonomous vehicles (AVs), commonly referred to as self-driving cars, are increasingly becoming a fundamental and recognizable part of our everyday lives. More than 50 companies, ranging from industry giants like Alphabet [12] and Tesla [10] to innovative startups such as WeRide [21] and Nuro [15], are actively developing AV technology. Many of these companies have already introduced their autonomous technologies to the market, including Waymo's robotaxi services [12], Tesla's Autopilot system [10], and Starship's robot delivery services [20].

These AVs are powered by autonomous driving software (ADS) that processes extensive data (e.g., road, weather, and obstacles) from their surroundings. This software employs various algorithms and machine learning techniques to perform tasks such as object recognition and navigation. Apollo [1] and Autoware [6] are among the most popular, production-grade or near production-grade, and open-source ADS, achieving a high level of autonomy (Level 4). However, like any technology, ADS can malfunction due to hardware issues or software bugs. These malfunctions can cause the AV to become immobile or even lose control, posing serious risks to public safety. Historical incidents highlight these dangers: In 2018, an Uber-operated

AV killed a pedestrian in Arizona [19]. In 2023, a GM Cruise robotaxi in San Francisco hit and dragged a woman around 20 feet after she was initially struck by a human-driven vehicle [11].

To ensure the safety and reliability of AVs, it is a common practice to conduct field operational tests. These tests evaluate the ADS by allowing the vehicle to operate autonomously under real-world conditions. However, real-world tests have significant limitations: they are costly and limited to specific geographical areas, potentially missing critical scenarios that could lead to accidents. In contrast, virtual testing, a valuable complement when field operational tests are unavailable, provides a controlled and safe environment by digitally simulating various driving scenarios. Modern high-fidelity simulators can also replicate conditions such as road and weather, thereby reducing the risks associated with real-world vehicular accidents.

Prior studies related to ADS testing techniques have predominantly conducted experiments only on Baidu Apollo [23] [25] [26] [27] [29] [30] [34] [35] [36] [37] [38] [41], with few on Autoware [28] or both [22] [24] [33] [39] [40]. This recurring choice is often attributed to the availability of tools developed specifically for Apollo, which inadvertently discourages the adaptation of these methods for other ADS like Autoware due to significant time constraints. Moreover, the architectural variations and evolutionary changes across different ADS platforms pose a challenge for the replicability and reusability of existing testing techniques. This situation underscores a crucial question: Are these techniques generalizable across various ADSes, or are they too dependent on the unique characteristics of a particular ADS?

Our research group has developed three ADS testing frameworks: scenoRITA [26], DoppelTest [27], and ConfVE [22]. Notably, we have already successfully adapted ConfVE [22] to Autoware, demonstrating the potential for broader applicability of our tools. This thesis specifically aims to expand on this success by adapting scenoRITA for use with Autoware. This endeavor explores the challenges of this migration, tests the generalizability of scenoRITA, and provides development support for testing Autoware.

More precisely, we transferred scenoRITA to Autoware, which can successfully generate and mutate driving scenarios that expose an ego car to safety and comfort scenarios for Autoware. During this migration, we identified the challenges of the re-implementation and developed some useful tools for testing Autoware. Our empirical experiments show that scenoRITA can generate test scenarios that trigger 63 unique safety and comfort violations in Autoware.

The organization of this thesis is summarized as follows: Chapter 2 introduces autonomous driving software and the testing framework scenoRITA. Chapter 3 identifies the challenges encountered during the re-implementation of scenoRITA for Autoware. Chapter 4 discusses the empirical evaluations and results obtained from these studies. Chapter 5 offers an in-depth discussion of the findings. Chapter 6 reviews the related work in the field. Finally, Chapter 7 concludes the thesis.

Chapter 2

Background

2.1 Overview of Autonomous Driving Software

2.1.1 Automation Levels

The Society of Automotive Engineers (SAE) categorizes driving automation into six levels, ranging from Level 0 (no automation) to Level 5 (complete automation) [17]. Table 2.1 outlines the degree of autonomy associated with each level. Levels 0 to 2 provide driving assistance support functionalities, while Levels 3 to 5 encompass fully automated driving capabilities. Apollo [1] and Autoware [6] are classified as Level 4 because they can operate without human intervention under certain conditions. In contrast, Tesla's Full Self-Driving (FSD) system [10], despite allowing drivers to disengage their hands from the steering wheel and feet from the pedals, remains classified as Level 2 since it still requires driver attentiveness.

Table 2.1: Automation Levels

Level	**Level Definition**	**Example Features**	**Driver Control**
0 (No Automation)	Warnings and momentary assistance	Emergency braking Lane departure warning	Yes
1 (Driver Assistance)	Steering or braking/acceleration assistance	Lane centering or Adaptive cruise control	Yes
2 (Additional Assistance)	Steering and braking/acceleration assistance	Lane centering and Adaptive cruise control	Yes
3 (Conditional Assistance)	Drive somewhere in limited conditions	Traffic jam support	When requested
4 (High Automation)	Drive somewhere in limited conditions	Robotaxi in limited conditions	Not necessary
5 (Full Automation)	Drive everywhere in all conditions	Robotaxi in all conditions	No

2.1.2 ADS Components

An ADS is a complex cyber-physical software system composed of multiple interconnected modules, each designed to handle specific functions essential for the operation of autonomous vehicles. The ADS includes six core modules: Routing, Localization, Perception, Prediction, Planning, and Control [29].

Routing develops high-level navigation strategies based on routing requests. **Localization** provides accurate information about the AV's kinematic features, such as position, orientation, speed, and acceleration. **Perception** utilizes data from sensors such as cameras, LiDAR, and radars to identify and classify stationary and moving objects within the traffic environment. **Prediction** leverages this data to analyze and anticipate the future paths of these objects. **Planning** utilizes the previously mentioned data to devise driving strategies and maneuvers, ensuring the AV navigates its environment safely and efficiently. Finally, **Control** takes these strategic plans and converts them into executable commands that operate the vehicle's functions, such as signaling turns and modulating speed.

2.2 Autoware

2.2.1 Overview

Autoware is a leading open-source software stack for AVs, built on the Robot Operating System (ROS). It provides essential functionalities for autonomous driving, such as localization, object detection, route planning, and vehicle control. Supported by a vibrant community and the Autoware Foundation, it benefits from contributions by various companies and researchers who provide development tools, application software, computing hardware, and sensors [6].

2.2.2 In Comparison with Apollo

To better understand the challenges of the transfer process, Table 2.2 offers a comparative analysis of Autoware and Apollo across four aspects relevant to the transfer process.

Table 2.2: Comparison between Autoware and Apollo

	Autoware	**Apollo**
Middleware Framework	ROS2	Cyber RT Framework
Map Format	Autoware Vector Map Autoware Point Cloud Map	Apollo HD Map
Simulator	CARLA LGSVL Scenario Simulator v2 AWSIM	CARLA LGSVL Sim-control
Simulation Record	ROS bag	Cyber Record

(1) Middleware Framework. The latest release of Autoware, Autoware Universe [6], is built on ROS2, leveraging its communication infrastructure and package management capabilities. In contrast, Apollo [1] has transitioned to its proprietary framework, Cyber RT, which is designed for enhanced robustness and performance.

(2) Map Format. Map representation varies significantly between Autoware and Apollo. Autoware uses two types of maps: the vector map, storing road semantic information in the Lanelet2 [32] format with additional modifications, and the point cloud map, providing geometric data in the Point Cloud Data (PCD) format. In contrast, Apollo uses the Apollo HD Map, a modified version of the ASAM OpenDRIVE format [3].

(3) Simulator. Several simulators are available for autonomous driving. CARLA [8], LGSVL [14], and AWSIM [7] offer game engine-based simulation environments, with AWSIM specifically supporting Autoware. The third-party simulators CARLA and LGSVL support both Autoware and Apollo. Additionally, Scenario Simulator v2 [18] and Sim-control [1] provide basic simulation features and native support for Autoware and Apollo, respectively.

(4) Record Format. Autoware records all messages produced during the simulation in the standard ROS bag format, while Apollo uses the Cyber Record format.

2.3 scenoRITA

scenoRITA [26] is a search-based scenario generation testing framework for ADS. It effectively creates valid test scenarios to uncover safety and comfort issues in ADS. scenoRITA employs five safety and comfort test oracles: collision, unsafe lane change (USLC), speeding, fast acceleration, and hard braking. The framework consists of five components: Scenario Generator, Generated Scenario Player, Planning Output Recorder, Grading Metrics Checker, and Duplicate Violations Detector.

Figure 2.1 illustrates the workflow of scenoRITA. Before launching scenoRITA, a set of domain-specific constraints is required to guide the Scenario Generator in producing valid driving scenarios. scenoRITA begins by generating several driving scenarios with randomly generated but valid obstacles under these constraints. The Generated Scenario Player converts these scenarios into inputs for the ADS to execute. During the simulation, the Planning Output Recorder captures and stores relevant messages. The Grading Metrics Checker evaluates each scenario against the five test oracles to identify any violations. The Scenario Generator calculates the fitness value for each scenario and evolves them using a genetic algorithm. This cycle of generation, execution, recording, evaluation, and evolution continues until the end of the process. Finally, the Duplicate Violations Detector reviews and removes duplicate violations, ensuring a unique set of safety and comfort violation scenarios.

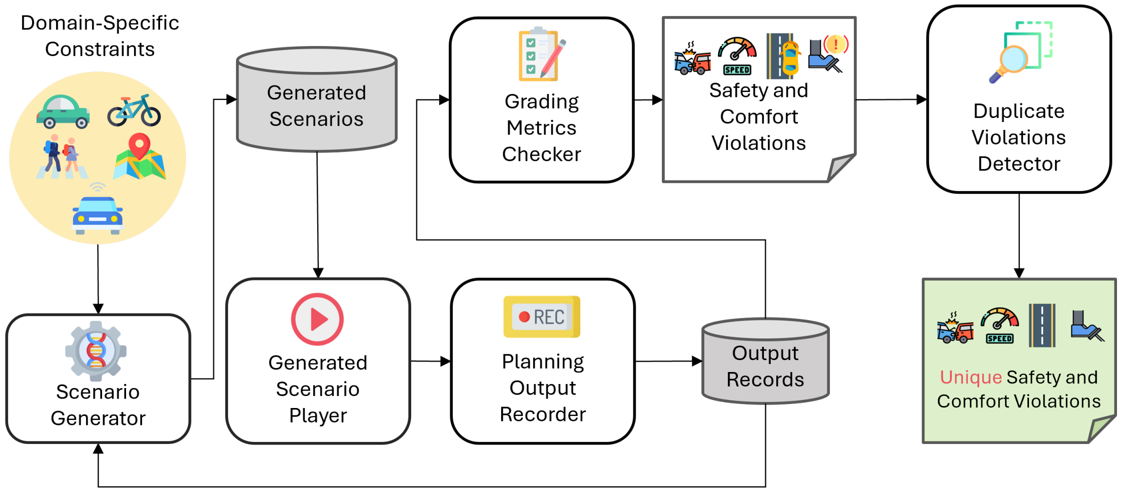

Domain-Specific Constraints
Generated Scenarios
Grading Metrics Checker
Safety and Comfort Violations
Duplicate Violations Detector
Scenario Generator
Generated Scenario Player
REC
Planning Output Recorder
Output Records
SPEED
Unique Safety and Comfort Violations

Figure 2.1: An Overview of scenoRITA [26]

Chapter 3

SCENORITA Transfer Challenges

This chapter examines the challenges of adapting scenoRITA for use with Autoware, focusing on domain-specific constraints, the Scenario Generator, the Generated Scenario Player, the Planning Output Recorder, the Grading Metrics Checker, and the Duplicate Violations Detector.

3.1 Domain-specific Constraints

Transferring domain-specific constraints is time-consuming, requiring careful consideration of various factors, including the ADS itself and simulators. Table 3.1 lists the requirements that the Scenario Generator must follow to ensure valid driving scenarios. In scenoRITA, we primarily consider two types of actors in a scenario: *ego car* and *obstacle*. The *ego car* actor has two attributes: initial position and final position. The *obstacle* actor has seven attributes: identifier, type, initial position, final position, speed, size, and motion. The primary challenges in transferring these constraints are related to **map formats** and **simulators**.

Table 3.1: Domain-specific Constraints

Actor	Attribute	Constraints
Ego Car	Initial Position (P_i)	$P_i \in Lane(vehicle)$
	Final Position (P_f)	$P_f \in Lane(vehicle) \wedge reachable(P_i, P_f, vehicle)$
Obstacle	ID (id_{obs})	$\forall o_1, o_2 \in obstacles(scenario), o_1 \neq o_2 \implies id_{o_1} \neq id_{o_2}$
	Type (T_{obs})	$T_{obs} \in \{vehicle,\ bicycle,\ pedestrian\}$
	Initial Position (P_i)	$P_i \in Lane(T_{obs})$
	Final Position (P_f)	$P_f \in Lane(T_{obs}) \wedge reachable(P_i, P_f, T_{obs})$
	Speed (v_{obs})	$v_{vehicle} \in [8, 110]\ km/h$ $v_{bicycle} \in [6, 30]\ km/h$ $v_{pedestrian} \in [4.5, 10.5]\ km/h$
	Size ($Z_{obs} = (length, width, height)$)	$Z_{car} = (4.0, 1.8, 2.5)\ m$ $Z_{bus} = (12.0, 2.5, 2.5)\ m$ $Z_{truck} = (8.4, 2.5, 2.5)\ m$ $Z_{motorcycle} = (2.2, 0.8, 2.5)\ m$ $Z_{bicycle} = (2.0, 0.8, 2.5)\ m$ $Z_{pedestrian} = (0.8, 0.8, 2.0)\ m$
	Motion (M_{obs})	$M_{obs} \in \{static,\ mobile\}$

3.1.1 Map Formats

Autoware vector map uses a modified version of the Lanelet2 map. In this vector map, each lane has a *subtype* tag specifying the lane type (e.g., road, highway, crosswalk). This tag determines which traffic participants (e.g., pedestrians, vehicles, bicycles) can use the lane. For instance, vehicles can use road lanes and highway lanes, while pedestrians can use crosswalk lanes. imposes additional constraints on the initial and final position attributes: (1) the initial and final positions of an actor must be within lanes permitted for its use, and (2) the final position must be reachable from the initial position based on the actor type. For example, a vehicle cannot have a valid routing path from a road lane to a crosswalk lane. In SCENORITA for Apollo, vehicles, bicycles, and pedestrians are treated the same and can be placed on any lane on the map.

3.1.2 Simulators

In this thesis, we used Scenario Simulator v2 for scenario execution. In an experiment, we created a scenario where an obstacle, classified as a *car* and measuring 13 meters in length, was placed directly in front of the ego car in the same lane. During the simulation, the ego car collided with the obstacle from behind, and there was no information about the obstacle in the perception messages. Upon further investigation, we discovered that Scenario Simulator v2 includes several built-in obstacle types that correspond to the participant types in the Lanelet2 framework (e.g., *bus*. *truck*, and *motorcycle*). When we changed the obstacle's type to *truck* or *bus*, the collision did not occur, and we were able to extract the obstacle information from the perception messages. This finding highlights the importance of matching obstacle size to the appropriate vehicle types in Scenario Simulator v2. Therefore, instead of generating a random size for each obstacle, we generate an obstacle with a random type and then assign its size attribute with a predefined value based on its type. A vehicle can be either a *motorcycle*, *car*, *truck*, or *bus*.

3.2 Scenario Generator

The Scenario Generator uses a genetic algorithm to create and evolve driving scenarios that identify safety and comfort violations. The genetic algorithm consists of three elements: genetic representation, fitness evaluation, and search operators. Transferring these elements is straightforward if all domain-specific constraints are identified and strictly followed. For example, Figure 3.1 illustrates the genetic representation of a driving scenario. Compared to the original paper, only the number of genes changes. We have kept the algorithms of this component unchanged, as it is the core of scenoRITA.

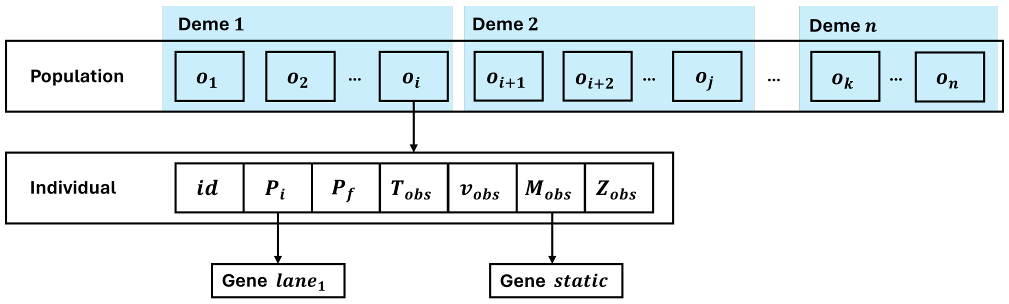


Figure 3.1: Genetic Representation of Tests in scenoRITA for Autoware

3.3 Generated Scenarios Player

The Generated Scenarios Player is tasked with transforming the genetic representation of scenarios from the Scenario Generator into inputs that are compatible with the ADS simulator and executing these scenarios. The primary challenges are (1) identifying the appropriate inputs for the simulator, and (2) accurately converting the genetic representation into these inputs.

In this work, the compatible input format for Scenario Simulator v2 is TIER IV Scenario Format v2 [18], a YAML-based format derived from ASAM OpenSCENARIO v1.2.0 [4]. To ensure precise conversion, we dedicated time to learning OpenSCENARIO [4] and analyzing existing scenario files from Autoware Evaluator [5] for any predefined configuration parameters.

3.4 Planning Output Recorder

The Planning Output Recorder is designed to capture messages generated during scenario execution. By setting the *record* parameter to *true* when launching the Generated Scenarios Player, the data produced during the simulation is stored in real-time in an SQLite database.

Autoware, by default, records nearly all topic messages during simulations, as shown in lines

2, 4, 5, and 6 of Figure 3.2. Our preliminary analysis revealed that an 80-second simulation with 10 actors could consume up to 3 GB of storage. Given our storage limitations, retaining records for thousands of scenarios is impractical. To address this challenge, we identified five essential ROS topics for analysis and modified the *openscenario_interpreter.cpp* file to record only this necessary data. This optimization, shown in Figure 3.3, selectively records the kinematic data of the ego car (lines 2 and 3), obstacle perception data (lines 4 and 5), and routing request data (line 6). As a result, we have reduced most simulation record files to under 50 MB, substantially increasing storage efficiency and reducing unnecessary I/O expenses.

```
1 record::start(
2    "-a",
3    "-o", boost::filesystem::path(osc_path).replace_extension("").string(),
4    "-x",
5    "/planning/scenario_planning/lane_driving/behavior_planning"
6    "/behavior_velocity_planner/debug/intersection"
7 );
```

Figure 3.2: Source Code L203-L210 of openscenario_interpreter.cpp (Reformatted)

```
1 record::start(
2    "/localization/acceleration",
3    "/localization/kinematic_state",
4    "/perception/object_recognition/objects",
5    "/perception/object_recognition/ground_truth/objects",
6    "/planning/mission_planning/route",
7    "-o", boost::filesystem::path(osc_path).replace_extension("").string()
8 );
```

Figure 3.3: Modified Version of openscenario_interpreter.cpp

3.5 Grading Metrics Checker

Upon completion or termination of the simulation, the Grading Metrics Checker assesses the current scenario for any comfort or safety violations. The primary challenges in transferring this component involve ensuring portability, analyzing simulation records, and evaluating scenarios against five test oracles.

3.5.1 Portability

Autoware is a large and complex software system with around 400 ROS packages. Building the entire latest Autoware project on a workstation with a 20-core CPU can take over two hours, making it impractical for low-performance laptops. However, for scenoRITA, compiling the entire project is only necessary for scenario simulation. Scenario generation and post-simulation analysis do not require a full build. For scenario generation, the only necessary packages are *lanelet2_extension* and *lanelet2_extension_python* to extract information from maps. For record analysis, the required packages are all ROS packages containing ROS messages or services used for communication between ROS nodes when running Autoware. These messages or services are essential for deserializing data from ROS bag files.

To facilitate the development and debugging of the Scenario Generator and the Grading Metrics Checker, we have identified a minimal set of ROS packages in Autoware. This minimal set includes the necessary packages for scenario generation and post-simulation analysis, which can be built in about 20 minutes on a Linux laptop with a 4-core CPU. Documentation for this reduced installation of Autoware is available on GitHub [16].

With this setup, scenoRITA for Autoware can be installed on any compatible Linux machine, enabling distributed scenario generation and output evaluation. This allows servers or workstations to focus solely on scenario execution, improving overall efficiency and performance.

3.5.2 Simulation Record Reader

An Apollo contributor developed the Python package *cyber_record* [9] for reading and extracting data from Cyber Record files used in previous ADS testing frameworks. To facilitate the transfer of ConfVE [22] and scenoRITA [26] to Autoware, we created a Python package for reading ROS bag files with APIs similar to *cyber_record*, minimizing the need for code refactoring.

3.5.3 Scenario Evaluation

Evaluating a scenario primarily involves identifying the relevant message topics that contain the data needed for analysis against the five test oracles. This challenge is mitigated when transferring the Planning Output Recorder. Another issue arises from perception noise. We discovered noise in the data from "/perception/object_recognition/objects" when retrieving obstacle information from perception messages. For example, an obstacle with a size of $Z_{obs} = (8.4, 2.5, 2.5)$ might contain both accurate information (ground truth) and noise, resulting in a reported size such as $Z_{obs} = (8.407060, 2.50004, 2.5)$. This noise can hinder the accurate evaluation of the distance between the ego car and surrounding obstacles. To address this, we modified the source code to publish the ground truth perception data and used that for record analysis.

3.6 Duplicate Violations Detector

The Duplicate Violations Detector is essential for identifying and eliminating duplicate scenarios, thereby minimizing the final set of violation scenarios. This component can be seamlessly integrated into Autoware by extracting the same features from the output of each scenario with violations, as is done with Apollo. This approach ensures that the process of identifying duplicates remains consistent and efficient across different ADSes.

Chapter 4

Empirical Studies

In order to empirically evaluate the technique implemented for Autoware, and to understand how differences affect scenarios and violations, we investigate the following research questions:

- **RQ1**: How effective are scenoRITA's generated driving scenarios at exposing Autoware to safety and comfort violations?
- **RQ2**: What is the runtime cost of scenoRITA for Autoware?
- **RQ3**: How diverse are scenarios generated by scenoRITA for Autoware?

4.1 Experiment Settings

The evaluation involved running 5,410 scenarios in a total of 46 hours on Autoware Universe v1.0 [6] using Scenario Simulator v2 [18]. We chose Scenario Simulator v2 because it (1) is developed by the Autoware team, (2) offers native support for Autoware, and (3) provides a basic UI without a game-like experience. This relationship between Scenario Simulator v2 and Autoware is similar to the one between Sim-control and Apollo. The experiments were performed on one machine running Ubuntu 22.04 equipped with a 20-core Intel(R) Core i7-12700KF, 32 GB RAM, and NVIDIA GeForce RTX 3090 Ti.

Table 4.1 presents details of the maps selected for our experiments. These four maps are sourced from Autoware Evaluator [5], an official component of the Autoware CI/CD pipeline. The Autoware developers use this tool for weekly regression testing as Autoware evolves. We selected maps of varying sizes for our experiments: Shalun (263 total lanes with 150 junction lanes), Hsinchu (788 total lanes with 83 junction lanes), AWF Virtual Map (764 total lanes with 288 junction lanes), and Nishi-Shinjuku (979 total lanes with 387 junction lanes).

Table 4.1: Information about Selected Maps

Map	**# Lanes**	**# Junction Lanes**	**Map File Size**
Shalun	263	150	844.5 KB
Hsinchu	788	83	1.6 MB
AWF Virtual Map (AVM)	764	288	7.7 MB
Nishi-Shinjuku (NS)	979	387	10.6 MB

For the configuration settings of search operators in the genetic algorithm, we followed the parameters specified in the original paper. Specifically, we used a crossover operator with a probability of 0.8 and applied mutations to individual scenarios with a probability of 0.2. The mutation process involved either adding a new obstacle from another scenario or removing an existing obstacle, each with a probability of 0.1.

We conducted seven experiments in total. For Experiment 1, 2, 3, 4, and 6, each generation consisted of 20 scenarios, while for Experiment 5 and 7, each generation had 30 scenarios. We chose to run five experiments with 20 scenarios per generation to ensure a manageable workload. The two experiments with 30 scenarios per generation were designed to test whether scenoRITA for Autoware could find more types of violations.

Scenario generation was stopped after 6 hours for Experiment 1, 2, 3, 4, and 6 to balance the need for extensive data collection with practical time constraints. For Experiment 5 and 7, we extended the generation time to 8 hours to observe the effects of prolonged scenario generation on the algorithm's effectiveness. Each scenario included 5 to 15 obstacles, and the maximum execution time was set to 60 seconds.

4.2 RQ1: Effectiveness of scenoRITA for Autoware

The effectiveness of scenoRITA for Autoware is demonstrated by its ability to produce safety and comfort violations. As shown in Table 4.2, scenoRITA detected 138 bug-revealing violations (before removing the duplicates) in Autoware. The patterns of these violations in Autoware are similar to those in Apollo, with the majority classified as unsafe lane change and hard braking, and fewer related to fast acceleration, speeding, and collision. After applying the Duplicate Violations Detector, 63 unique violations were identified.

Table 4.2: Bug-revealing Violations Found by scenoRITA on Autoware

Exp.	**Map**	**Speeding**	**USLC**	**Hard Braking**	**Fast Acceleration**	**Collision**
1	Shalun	0	12	3	0	0
2	Hsinchu	0	0	10	0	0
3	NS	0	10	0	0	0
4	NS	**1**	3	2	0	0
5	NS	0	14	8	0	0
6	AVM	0	9	0	0	0
7	AVM	0	34	28	0	**4**

Notably, there are four collision violations in Experiment 7. Figure 4.1 illustrates a collision at an intersection, showing two vehicles represented as rectangles: the obstacle vehicle in red

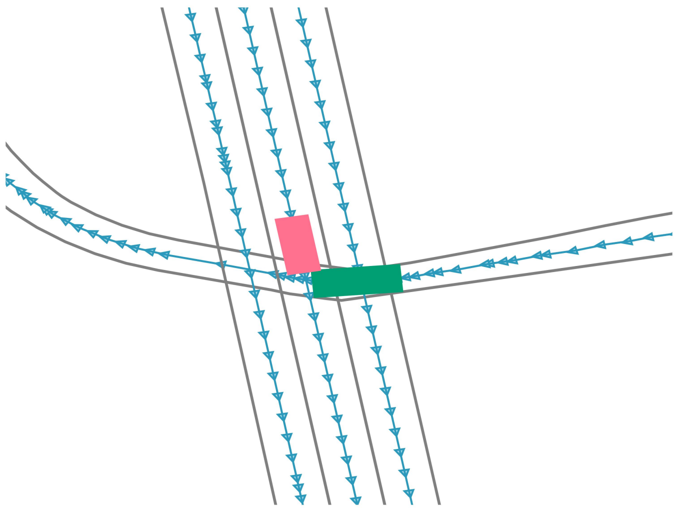


Figure 4.1: A Collision at an Intersection

and the ego car in green. The gray lines indicate the lane boundaries, and the arrow lines within the lane boundaries show the direction of traffic. The ego car is attempting to cross the intersection and make a right turn, while the obstacle vehicle is traveling straight along the road that conflicts with the ego car's lane at a speed of 3.65 m/s. The collision at the intersection highlights a critical failure in collision avoidance.

Upon further examination of the collision, we found that while the Prediction module functioned correctly, predicting three possible future paths based on the obstacle's current location, the Planning module failed to respond appropriately when these predicted paths intersected with the ego car's planned trajectory. In our scenario, we did not configure the traffic lights, leading both the ego car and the obstacle to assume they had a green light to proceed through the intersection. According to the documentation of the Planning module [13], the obstacle was in a yield lane outside the ego car's attention area. Consequently, the ego car did not react to the obstacle, even though their paths intersected. It appears that Autoware was unable to handle this emergency situation. In contrast, Apollo is likely to handle such situations if a vehicle runs a red light [2].

4.3 RQ2: Runtime Cost of scenoRITA for Autoware

Table 4.3 provides a detailed breakdown of runtime metrics across seven experimental setups of scenoRITA for Autoware. We calculate the average time cost both per scenario and per generation, with end-to-end generation times highlighting the overall efficiency of scenoRITA for each experiment. For instance, in Experiment 1, it took an average of 0.21 seconds to generate a scenario, 66.77 seconds to execute a scenario, 62.43 seconds to analyze a simulation record, and 129.41 seconds for the entire end-to-end process for a scenario. For each generation, it took an average of 1.46 seconds to apply the crossover/mutation operators (Cx/Mut), 545.28 seconds to evaluate all scenarios, and 0.00 seconds to select scenarios for the next generation. Additionally, we observed that scenario generation and crossover/mutation operations are significantly shorter compared to scenario execution and generation evaluation times, respectively.

We found that the scenario generation time did not correlate with map complexity and remained consistent across different maps. This consistency could be due to the limited search space for generating pedestrians and bicycles, as they were restricted to designated lanes. In contrast, Apollo allows pedestrians, bicycles, and vehicles to use all lanes.

We also discovered that some scenarios failed to execute due to performance issues, as simultaneous scenario execution and analysis consumed a significant amount of CPU resources. This detailed breakdown is crucial for identifying performance bottlenecks and optimizing the scenoRITA framework for better efficiency when integrated with Autoware.

This issue can be mitigated by separating the execution and analysis processes onto different machines. As mentioned in Chapter 3.5.1, we provided reduced installation support for Autoware. This enhancement allows us to distribute the analysis processes across multiple servers, thereby avoiding resource contention and improving overall system efficiency. By decoupling these tasks, we can ensure that execution processes run smoothly without being hindered by the resource demands of analysis, leading to more reliable and faster performance.

Table 4.3: Runtime Cost (in seconds) of scenoRITA for Autoware

Exp.	**Stats per Scenario**				**Stats per Generation**			
	Generate	**Play**	**Analyze**	**E2E**	**Mut/Cx**	**Evaluate**	**Select**	**E2E**
1	0.21	66.77	62.43	129.41	1.46	545.28	0.00	546.74
2	0.27	58.17	99.42	157.85	7.66	649.81	0.00	657.47
3	0.31	64.82	27.51	92.64	1.02	488.52	0.02	489.57
4	0.13	55.26	99.31	154.70	3.58	634.18	0.00	637.76
5	0.29	67.52	142.01	209.81	6.13	776.73	0.00	782.82
6	0.24	55.09	82.95	138.28	9.25	788.92	0.00	798.17
7	0.23	68.40	145.21	213.84	6.96	1,047.83	0.00	1,054.75

4.4 RQ3: Diversity of the Generated Scenarios

Due to performance issues with our workstation, some scenarios could not be executed. Instead, we shifted our focus to evaluating the diversity of the executable scenarios. Since there is no well-established method to evaluate the diversity of generated scenarios, we carried out both qualitative and quantitative analyses of the scenarios.

We first visualized the map coverage for Experiment 1, 2, 3, and 6, as they had the identical experimental setups. Figure 4.2 displays the map coverage results for each experiment. The blue markings indicate the total ego car trajectories in each experiment, while the red markings represent the distribution of the obstacles' trajectories. We observed that within a 6-hour experiment, scenoRITA for Autoware was able to produce diverse scenarios, with ego cars and obstacles traveling through different sections of the map. Although we tried to determine the exact proportion of the map covered by the ego car's total trajectories, the calculations required were too complex to complete.

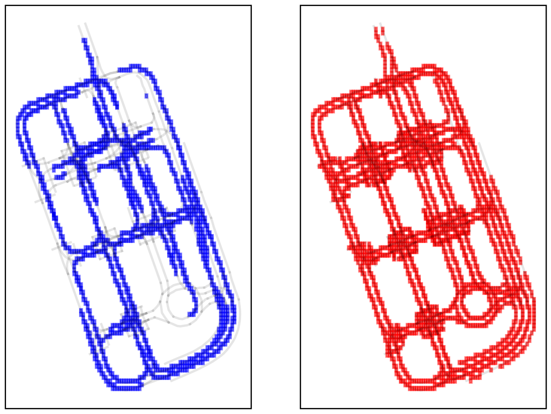
Second, we approximated scenario diversity using the diversity of the ego car's trajectories, excluding the trajectories of obstacles for simplicity. Two scenarios are considered the same if, at any timestamp t (with $t_0 = 0$ marking the start of the ego car's movement), the Euclidean distance between the locations of the two ego cars does not exceed the threshold $\theta = 1\ m$. This threshold is based on the default value used by Autoware Evaluator [5]. Table 4.4 illustrates the diversity of executable scenarios across four experiments. The diversity percentages are as follows: 60.38% for Experiment 1, 59.97% for Experiment 2, 86.59% for Experiment 3, and 86.53% for Experiment 6. The higher diversity in Experiments 3 and 6 is primarily due to the increased complexity of the NS and AVM maps, leading to more failed scenarios compared to the Shalun and Hsinchu maps.

Table 4.4: Diversity of the Executable Scenarios for Four Experiments

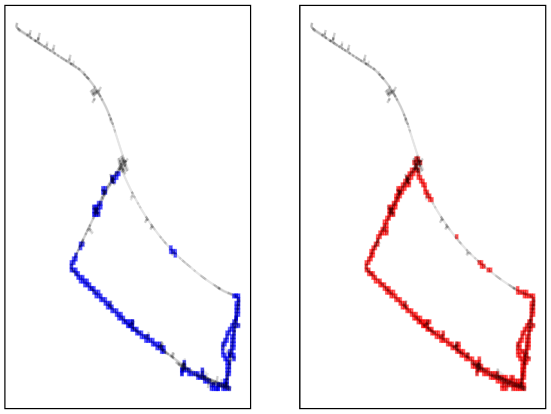
Exp.	**Map**	**Unique**	**Total**	**Diversity**
1	Shalun	413	684	60.38%
2	Hsinchu	367	612	59.97%
3	NS	142	164	86.59%
6	AVM	257	297	86.53%

In the process of examining map coverage, we identified that certain lanes were disconnected from the rest of the map. For instance, Figure 4.3 depicts a segment of the Hsinchu map. Based on the map's layout, Lanes 1720 and 1721, which are successors to Lane 1718, should be reachable from it. However, in a specific test scenario where the initial position of an ego car was set on Lane 1718 and the final position on Lane 1720, the scenario could not be executed successfully. A detailed inspection of the map data exposed significant misalignments in the lane boundaries: the left boundary of the start of Lane 1720 does not align with the end of the left boundary of Lane 1718, and a similar misalignment is present in the right boundaries of these lanes. These differences prevent the ego car from moving from Lane 1718 to Lane 1720, causing the scenario to fail.

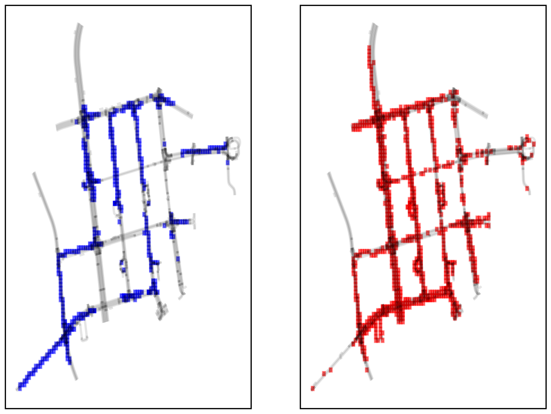
After pinpointing these misalignments, minor adjustments were made to rectify the boundary alignments: the end point of the left boundary of Lane 1718 was added to the left boundary of Lane 1720 as its starting point, with a similar correction applied to the right boundary. Following these adjustments, the ego car could successfully travel from Lane 1718 to Lane 1720.



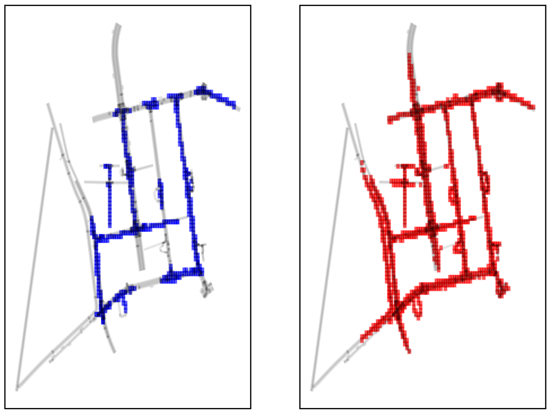
(a) Exp. 1: Shalun



(b) Exp. 2: Hsinchu



(c) Exp. 3: Nishi-Shinjuku



(d) Exp. 6: AWF Virtual Map

Figure 4.2: Map Coverage Visualization for Four Experiments. The blue markings indicate the total ego car's trajectory in each experiment, while the red markings represent the distribution of the obstacles' trajectories.

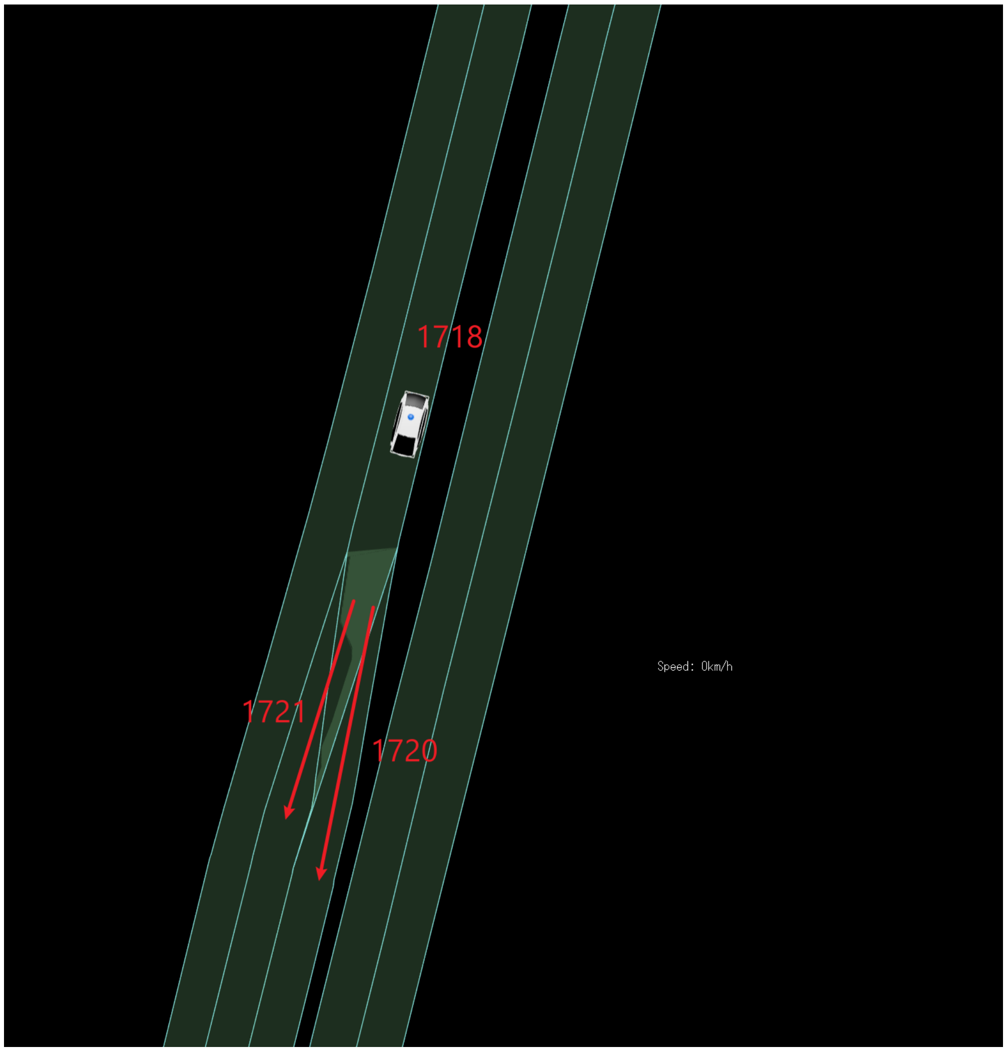


Figure 4.3: A Section of the Hsinchu Map: Lane 1720 and Lane 1721 follow Lane 1718, but only Lane 1721 is accessible from Lane 1718.

Chapter 5

Discussion

5.1 Threats to Validity

Internal Threats. One potential internal threat is the selection of scenario preparation and execution times. As pioneers in conducting experiments on Autoware Universe v1.0, we had limited references for these selections. Therefore, we adhered to the configuration settings outlined in our recently accepted ConfVE [22] paper.

Another internal threat is setting the dimensions of one type of obstacle to a fixed value, which could reduce the variety of obstacles. This issue can be mitigated by using four distinct vehicle instances: *car*, *bus*, *truck*, and *motorcycle*, each with different lengths and widths. These vehicle types can effectively represent most obstacles encountered in daily life.

Additionally, the reliability of the simulator is a potential internal threat. We chose to use Scenario Simulator v2 to execute scenarios instead of alternatives like CARLA or AWSIM. Autoware developers conduct weekly regression testing using Scenario Simulator v2, which we believe helps mitigate this threat.

External Threats. One external threat is that we applied scenoRITA to a specific version of Autoware. To mitigate this threat, we chose a publicly stable release version, v1.0, instead of using the main branch or monthly updated versions.

5.2 Rule-based ADS

As for now, Apollo and Autoware are rule-based autonomous driving software. However, it is not the permanent solution for autonomous vehicles since it relies too heavily on HD maps. Instead, deep learning-based approaches are emerging as the preferred solution due to their ability to learn and improve autonomously. However, this transition presents a significant challenge in testing autonomous driving software. Deep learning systems, being adaptive and data-driven, complicate the task of ensuring safety and reliability. Traditional testing methods may be inadequate, necessitating the development of new testing frameworks that can handle the dynamic and evolving nature of deep learning models. This includes advancements in scenario-based testing, simulation environments, and real-world validation techniques to ensure these autonomous systems perform safely and effectively across diverse and unpredictable conditions.

Chapter 6

Related Work

Numerous studies on ADS testing have demonstrated the effectiveness of their tools, predominantly utilizing **Apollo** as the primary testing platform. AV-FUZZER [29] is a fuzzing ADS testing framework, which employs collision metrics to guide the generation and selection of scenarios. sCENoRITA [26] is the core reference of this study. MOSAT [37] aims to efficiently generate adversarial and diverse safety-critical scenarios using a multi-objective genetic algorithm to expose various safety violations in ADSes. CRISCO [38] is a technique to efficiently generate safety-critical test scenarios for ADS by mining influential behavior patterns from real traffic trajectories. DeepCollision [30] employs a reinforcement learning approach to enhance scenario generation. DoppelTest [27] is a novel framework that operates each vehicle via ADS to efficiently detect system bugs. AutoFuzz [41] integrates a neural network into its fuzzing engine to refine the generation of test scenarios that focus on collisions and lane deviations. BehAVExplor [23] is a diversity and violation-guided fuzzing framework to generate critical scenarios with diverse behaviors of ADS. STRaP [25] focuses on reducing test suites by slicing and segmenting scenario records. SPECTRE [31] concentrates on selecting and prioritizing test suites to accommodate the evolution of ADS. LawBreaker [35] is an ADS testing framework that uses a driver-oriented specification language and a fuzzing engine to test ADS against real-world traffic laws. REDriver [36] is a modular runtime enforcement framework for ADS that uses Signal Temporal Logic to enforce safety by monitoring and minimally adjusting planned trajectories according to national traffic laws. ACAV [34] is a framework that automates the causality analysis of AV accident recordings to identify critical safety-related events.

Other research includes experiments on **Autoware or both platforms**. DriveFuzz [28] utilizes four types of mutators along with a driving quality feedback engine to guide test mutation and selection. PlanFuzz [40] is a novel dynamic testing tool for discovering semantic DoS vulnerability in AD behavioral planning. ACERO [33] is a framework for discovering adversarial driving maneuvers that cause AV to violate safety rules while appearing benign and maintaining low liability for the attacker. SCTrans [24] transforms realistic driving records into virtual test scenarios. CONFVE [22] explores emergent failures in ADS by fuzzing configuration options. Additionally, Underwood et al. [39] propose a metamorphic testing framework aimed at evaluating the non-deterministic behaviors of test outcomes and generating more complex driving scenarios.

Chapter 7

Conclusion

In conclusion, this study has demonstrated the feasibility and challenges of transferring our ADS testing technique, scenoRITA, to a different autonomous driving system, Autoware. The success of re-implementation also confirms the generalizability of scenoRITA.

Through the successful migration of the framework, we have shown that it is possible to overcome the challenges posed by the different architecture and implementations inherent in different ADSes. The identified challenges of the research pave the way for more universal and effective approaches in the field of autonomous driving.

Our empirical experiments show that scenoRITA for Autoware can identify 63 unique violations across four types of safety and comfort oracles. We also uncovered potential bugs in the Planning module and one map from Autoware Evaluator. We offer several technical supports for testing and debugging of Autoware, including a reduced installation of Autoware and a simulation record reader.

Bibliography

[1] Apollo: An open autonomous driving platform. https://bit.ly/3UQ7sPI, 2024.

[2] Apollo Issue #8559: why need traffic_light_unprotected_left(right)_turn_scenario. https://bit.ly/3X14dYe, 2024.

[3] ASAM OpenDRIVE. https://bit.ly/4bS35dF, 2024.

[4] ASAM OpenSCENARIO: User Guide. https://bit.ly/3yyBgcg, 2024.

[5] Autoware Evaluator. https://evaluation.tier4.jp/, 2024.

[6] Autoware: the world's leading open-source software project for autonomous driving. https://bit.ly/3xPVmuH, 2024.

[7] AWSIM. https://github.com/tier4/AWSIM, 2024.

[8] CARLA Simulator. https://carla.org/, 2024.

[9] cyber_record: cyber_record offline parse tool. https://bit.ly/44YJXsl, 2024.

[10] Full Self-Driving (Beta). https://bit.ly/4b34RZk, 2024.

[11] GM's Cruise Loses Its Self-Driving License in San Francisco After a Robotaxi Dragged a Person. https://bit.ly/3yEf0xq, 2024.

[12] Google's Self-Driving Car Caused Its First Accident. https://bit.ly/3acQwgO, 2024.

[13] Intersection - Autoware Universe Documentation. https://bit.ly/3WOSQ5Q, 2024.

[14] LGSVL. https://github.com/lgsvl/simulator, 2024.

[15] Nuro. https://www.nuro.ai, 2024.

[16] Reduced Installation of Autoware for SCENORITA. https://bit.ly/3VcJAau, 2024.

[17] SAE Levels of Driving Automation Refined for Clarity and International Audience. https://bit.ly/3VdpAUU, 2024.

[18] Scenario Testing Framework for Autoware. https://bit.ly/3RkEBCn, 2024.

[19] Self-Driving Uber Car Kills Pedestrian in Arizona, Where Robots Roam. `https://nyti.ms/40IHWxd`, 2024.

[20] Starship Technologies: Autonomous robot delivery. `https://www.starship.xyz/`, 2024.

[21] WeRide. `https://www.weride.ai`, 2024.

[22] Y. Chen, Y. Huai, S. Li, C. Hong, and J. Garcia. Misconfiguration Software Testing for Failure Emergence in Autonomous Driving Systems. In *Proceedings of the 32nd ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2024, 2024. To appear.

[23] M. Cheng, Y. Zhou, and X. Xie. BehAVExplor: Behavior Diversity Guided Testing for Autonomous Driving Systems. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2023, page 488–500, New York, NY, USA, 2023. Association for Computing Machinery.

[24] J. Dai, B. Gao, M. Luo, Z. Huang, Z. Li, Y. Zhang, and M. Yang. SCTrans: Constructing a Large Public Scenario Dataset for Simulation Testing of Autonomous Driving Systems. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024. Association for Computing Machinery.

[25] Y. Deng, X. Zheng, M. Zhang, G. Lou, and T. Zhang. Scenario-Based Test Reduction and Prioritization for Multi-Module Autonomous Driving Systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2022, page 82–93, New York, NY, USA, 2022. Association for Computing Machinery.

[26] Y. Huai, S. Almanee, Y. Chen, X. Wu, Q. A. Chen, and J. Garcia. SCENORITA: Generating Diverse, Fully Mutable, Test Scenarios for Autonomous Vehicle Planning. *IEEE Transactions on Software Engineering*, 49(10):4656–4676, 2023.

[27] Y. Huai, Y. Chen, S. Almanee, T. Ngo, X. Liao, Z. Wan, Q. A. Chen, and J. Garcia. Doppelgänger Test Generation for Revealing Bugs in Autonomous Driving Software. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 2591–2603, 2023.

[28] S. Kim, M. Liu, J. J. Rhee, Y. Jeon, Y. Kwon, and C. H. Kim. DriveFuzz: Discovering Autonomous Driving Bugs through Driving Quality-Guided Fuzzing. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 1753–1767, New York, NY, USA, 2022. Association for Computing Machinery.

[29] G. Li, Y. Li, S. Jha, T. Tsai, M. Sullivan, S. K. S. Hari, Z. Kalbarczyk, and R. Iyer. AV-FUZZER: Finding Safety Violations in Autonomous Driving Systems. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pages 25–36, 2020.

[30] C. Lu, Y. Shi, H. Zhang, M. Zhang, T. Wang, T. Yue, and S. Ali. Learning Configurations of Operating Environment of Autonomous Vehicles to Maximize their Collisions. *IEEE Transactions on Software Engineering*, 49(1):384–402, 2023.

[31] C. Lu, H. Zhang, T. Yue, and S. Ali. Search-Based Selection and Prioritization of Test Scenarios for Autonomous Driving Systems. In U.-M. O'Reilly and X. Devroey, editors, *Search-Based Software Engineering*, pages 41–55, Cham, 2021. Springer International Publishing.

[32] F. Poggenhans, J.-H. Pauls, J. Janosovits, S. Orf, M. Naumann, F. Kuhnt, and M. Mayr. Lanelet2: A high-definition map framework for the future of automated driving. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 1672–1679, 2018.

[33] R. Song, M. O. Ozmen, H. Kim, R. Muller, Z. B. Celik, and A. Bianchi. Discovering Adversarial Driving Maneuvers against Autonomous Vehicles. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 2957–2974, Anaheim, CA, Aug. 2023. USENIX Association.

[34] H. Sun, C. M. Poskitt, Y. Sun, J. Sun, and Y. Chen. ACAV: A Framework for Automatic Causality Analysis in Autonomous Vehicle Accident Recordings. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024. Association for Computing Machinery.

[35] Y. Sun, C. M. Poskitt, J. Sun, Y. Chen, and Z. Yang. LawBreaker: An Approach for Specifying Traffic Laws and Fuzzing Autonomous Vehicles. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, New York, NY, USA, 2023. Association for Computing Machinery.

[36] Y. Sun, C. M. Poskitt, X. Zhang, and J. Sun. REDriver: Runtime Enforcement for Autonomous Vehicles. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024. Association for Computing Machinery.

[37] H. Tian, Y. Jiang, G. Wu, J. Yan, J. Wei, W. Chen, S. Li, and D. Ye. 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*, ESEC/FSE 2022, page 94–106, New York, NY, USA, 2022. Association for Computing Machinery.

[38] H. Tian, G. Wu, J. Yan, Y. Jiang, J. Wei, W. Chen, S. Li, and D. Ye. 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*, ASE '22, New York, NY, USA, 2023. Association for Computing Machinery.

[39] R. Underwood, Q.-H. Luu, and H. Liu. A Metamorphic Testing Framework and Toolkit for Modular Automated Driving Systems. In *2023 IEEE/ACM 8th International Workshop on Metamorphic Testing (MET)*, pages 17–24, 2023.

[40] Z. Wan, J. Shen, J. Chuang, X. Xia, J. Garcia, J. Ma, and Q. A. Chen. Too Afraid to Drive: Systematic Discovery of Semantic DoS Vulnerability in Autonomous Driving Planning under Physical-World Attacks. The Internet Society, 2022.

[41] Z. Zhong, G. Kaiser, and B. Ray. Neural Network Guided Evolutionary Fuzzing for Finding Traffic Violations of Autonomous Vehicles. *IEEE Trans. Softw. Eng.*, 49(4):1860–1875, Apr 2023.