SCENORITA: Generating Diverse, Fully Mutable, Test Scenarios for Autonomous Vehicle Planning

Yuqi Huai, Sumaya Almanee, Yuntianyi Chen, Xiafa Wu, Qi Alfred Chen, and Joshua Garcia, *Member, IEEE*

***Abstract*—Autonomous Vehicles (AVs) leverage advanced sensing and networking technologies (e.g., camera, LiDAR, RADAR, GPS, DSRC, 5G, etc.) to enable safe and efficient driving without human drivers. Although still in its infancy, AV technology is becoming increasingly common and could radically transform our transportation system and by extension, our economy and society. As a result, there is tremendous global enthusiasm for research, development, and deployment of AVs, e.g., self-driving taxis and trucks from Waymo and Baidu. The current practice for testing AVs uses virtual tests—where AVs are tested in software simulations—since they offer a more efficient and safer alternative compared to field operational tests. Specifically, search-based approaches are used to find particularly critical situations. These approaches provide an opportunity to automatically generate tests; however, systematically creating *valid* and *effective* tests for AV software remains a major challenge. To address this challenge, we introduce SCENORITA, a test generation approach for AVs that uses an evolutionary algorithm with (1) a novel gene representation that allows obstacles to be *fully mutable*, hence, resulting in more reported violations and more diverse scenarios, (2) 5 test oracles to determine both safety and motion sickness-inducing violations and (3) a novel technique to identify and eliminate duplicate tests. Our extensive evaluation shows that SCENORITA can produce test scenarios that are more effective in revealing ADS bugs and more diverse in covering different parts of the map compared to other state-of-the-art test generation approaches.**

***Index Terms*—Embedded/cyber-physical systems, search-based software engineering, software testing.**

I. Introduction

AUTONOMOUS vehicles (AVs), a.k.a. self-driving cars, are becoming a pervasive and ubiquitous part of our daily life. More than 50 corporations are actively working on AVs, including large companies such as Google's parent company Alphabet, Ford, and Intel [1], [2], [3]. Some of these companies (e.g., Alphabet's Waymo, Lyft, and Baidu) are already serving customers on public roads [4], [5], [6]. Experts forecast that AVs will drastically impact society, particularly by reducing accidents [7]. However, crashes caused by AVs indicate that achieving this lofty goal remains an open challenge. Despite the fact that companies such as Tesla [8], Waymo [1], or Uber [9] have released prototypes of AVs with a high level of autonomy, they have caused injuries or even fatal accidents to pedestrians. For instance, an AV of Uber killed a pedestrian in Arizona back in 2018 [10]. AVs with lower levels of autonomy have resulted in more fatalities in recent years [10], [11], [12], [13], [14], [15], [16], [17].

Prior research has revealed a lack of standardized procedures to test AVs [18] and the inability of current approaches to effectively translate traditional software testing approaches into the space of AVs [19], [20]. A common practice for testing AV software lies in field operational tests, in which AVs are left to drive freely in the physical world. This approach is not only expensive and dangerous, but also ineffective since it misses critical testing scenarios [21] Virtual tests, where AVs are tested in software simulations, offer a far more efficient and safer alternative. While these tests provide an opportunity to automatically generate tests, they come with the key challenge of *systematically generating scenarios which expose AVs to safety-critical and motion sickness-inducing situations*.

To address this challenge, we propose SCENORITA (**scen**ario Gene**R**at**I**on **T**esting for **A**Vs), a test generation framework which aims to find safety and motion sickness-inducing violations in the presence of an evolving traffic environment. SCENORITA combines both (i) AV software domain knowledge and (ii) search-based testing [22], [23]. These two elements have been combined by previous techniques to test AVs by automatically generating safety-critical scenarios [24], [25], [26], [27], [28], [29], [30]. However, unlike these approaches, SCENORITA's gene representation enables obstacles to be *fully mutable*, i.e., changing any of the properties such as its start and end location, type (e.g., vehicle, pedestrian, and bike), speed, size, and mobility (e.g., static or dynamic) can be altered. Previous techniques do not specify their gene representations or do so in such a way that allows obstacles to be only *partially mutable*: obstacles' attributes are altered only during mutation and with a small probability, while during crossover, obstacles are transferred across scenarios without altering their states or properties [24], [25], [26], [30], [31]. Thus, these techniques ignore the challenge of ensuring the creation of valid obstacle trajectories, reducing their *effectiveness at generating driving scenarios with unique violations*.

Manuscript received 27 March 2022; revised 13 August 2023; accepted 15 August 2023. Date of publication 30 August 2023; date of current version 17 October 2023. This work was supported by NSF under Grants 1823262, 1929771, 1932464, and 2145493. Recommended for acceptance by D. Bianculli. *(Yuqi Huai and Sumaya Almanee contributed equally to this work.) (Corresponding author: Joshua Garcia.)*

The authors are with the University of California Irvine, Irvine, CA 92697 USA (e-mail: yhuai@uci.edu; salmanee@uci.edu; yuntianc@uci.edu; xiafaw@uci.edu; alfchen@uci.edu; joshug4@uci.edu).

Digital Object Identifier 10.1109/TSE.2023.3309610

Other limitations of prior work include generating driving scenarios with: (i) manual setup and a limited number of scenario types; (ii) a small fixed number of obstacles per scenario; and (iii) obstacles with fixed trajectories and limited maneuvers which require manual specification. SCENORITA's gene representation allows obstacles to fully evolve throughout the test generation process, while still adhering to traffic laws and providing the ego car with ample amount of time to react to any potential violations. This novel gene representation addresses the limitations in the state-of-the-art approaches (Section II), as follows:

- SCENORITA eliminates the need for manual and fixed scenario setup by doing the following: (i) given any HD map, SCENORITA parses the map and generates a directed graph of all points (nodes) residing in the map and their connected lanes (edges), (ii) this directed graph is used to validate the trajectories of ego car and obstacles at the start of the test generation, and as they evolve and mutate during the evolution process. In other words, there is no need to manually set a fixed trajectory for the ego car or obstacles, as SCENORITA handles that automatically.
- Since there is no need to manually setup any scenarios (i.e., obstacles and the ego car can be placed anywhere in the map and it is ensured that they adhere to traffic laws and are within acceptable proximity), this allows SCENORITA to generate a much larger, diverse, and complex set of scenario types which: (i) cover as many lanes and lane types, in a map, as possible (e.g., single- or multi-lane roads with either the same or the opposite traffic direction, U-turns, roundabouts, cross or T intersections, merged lanes, etc.); (ii) does not have a limited or fixed number of obstacles; (iii) supports any combination of maneuvers per scenario for both an ego car and an obstacle. For example, in one scenario, an ego car can be directed to follow an obstacle before it changes lanes and then turns right at an intersection.
- As shown later in our evaluation results, SCENORITA is more effective, more diverse, and more efficient compared to the state-of-the-art approaches (i.e., AV-FUZZER and AutoFuzz) that, unlike SCENORITA, require significant manual specification per scenario type, especially for trajectories.

Additionally, previous work on AV software testing uses a highly limited number of test oracles for ensuring safety and no oracles for assessing motion sickness-inducing movement of an AV: State-of-the-art AV testing approaches (*AC3R* [31], *AsFault* [25], AV-FUZZER [24], AutoFuzz [30] and Abdessalem et al. [28], [29]) use only two oracles for checking if (1) the ego car reaches its final expected position while avoiding a crash (i.e., collision detection) and (2) if a vehicle drives off the road (i.e., off-road detection). As a result, the fitness functions these techniques utilize are overly simplified—substantially reducing their degree of safety assurance while completely ignoring rider comfort and motion sickness. Research has shown that a rider's discomfort increases when a human is a passenger rather than a driver—with up to one-third of Americans experiencing motion sickness, according to the National Institutes of Health (NIH) [32], [33], [34].

To overcome such limitations, SCENORITA utilizes 5 test oracles (i.e., collision detection, speeding detection, unsafe lane change, fast acceleration, and hard braking) and corresponding fitness functions—which are based on grading metrics for driving behavior defined by Apollo's developers [35]. Apollo is a high autonomy (i.e., Level 4), open-source, production-grade AV software system created by Baidu. The Society of Automotive Engineers (SAE) defines 6 levels of vehicle autonomy [36], [37], where Level 4 (L4) AV systems, such as Apollo, have the AV perform all driving functionality under certain circumstances, although human override is still an option. Apollo is selected by Udacity to teach state-of-the-art AV technology [38] and can be directly deployed on real-world AVs such as Lincoln MKZ, Lexus RX 450h, GAC GE3, and others [39], [40], and has mass production agreements with Volvo and Ford [41]. Additionally, Apollo has already started serving the general public in cities (e.g., a robo-taxi service in Changsha, China [42]).

The main contributions of this paper are as follows:

- We introduce SCENORITA, a search-based testing framework, with a novel gene representation and *domain-specific constraints*, that automatically generates *valid* and *effective* driving scenarios. SCENORITA aims to maximize the number of scenarios with unique violations and relies on a novel gene representation of driving scenarios, which enables the search to be more *effective*: Our gene representation allows the genetic algorithm to alter the states and properties of obstacles in a scenario, allowing them to be fully mutable. Moreover, this representation enables SCENORITA to generate diverse scenario types with unlimited vehicle maneuvers and without the need for manual setup. We specify a set of *domain-specific constraints* to ensure that the generated driving scenarios are *valid*. To the best of our knowledge, we are the first to define the exact values of these constraints, which are obtained from authoritative sources such as the National Center for Health Statistics, Federal Highway Administration, and the US Department of Transportation [43], [44], [45], [46].
- To improve the effectiveness of SCENORITA, we automate the process of identifying and eliminating duplicate violations by using an unsupervised clustering technique to group driving scenarios, with similar violations, according to specific features.
- We utilize 5 test oracles and corresponding fitness functions to assess different aspects of AVs—ranging from traffic and road safety (i.e., collision detection, speeding detection, and unsafe lane change) to a rider's comfort (i.e., fast acceleration and hard braking). To the best of our knowledge, SCENORITA is the first search-based testing technique for AV software that uses multiple test oracles at the same time and considers both comfort and safety violations as part of those oracles.
- We evaluate the effectiveness and efficiency of SCENORITA by comparing it against state-of-the-art approaches (i.e., AV-FUZZER and AutoFuzz) that do

not utilize a fully-mutable gene representation and thus, unlike SCENORITA, require significant manual effort to specify each scenario type, especially for trajectories.

Our extensive evaluation—which consists of executing a total of 79,051 virtual tests on Baidu Apollo using 4 high-definition maps of cities/street blocks located in California: Borregas Ave (60 lanes); San Mateo (1,305 lanes); San Francisco (1,524 lanes); and Sunnyvale (3,061 lanes)—shows that SCENORITA generates driving scenarios that expose the ego car to critical and realistic situations. SCENORITA found 1,146 *unique* comfort and safety violations, while 2 state-of-the-art approaches found a total of 7 and 37, respectively. Furthermore, SCENORITA generated between 4.95 and 8.22 times more scenarios than those approaches in the same amount of time. We make our testing framework available online to ensure reusability, reproducibility, and others to build upon our work [47].

II. Related Work

A wide array of studies focus on **applying traditional testing techniques to AVs** including adaptive stress testing [48], where noise is injected into the input sensors of an AV to cause accidents; fitness function templates for testing automated and autonomous driving systems with heuristic search [49]; and search-based optimization [50]. These studies provide limited insights into the testing of real-world AVs, since they do not evaluate their techniques on open-source, production-grade AV software.

Other related work focuses on the **vision and machine-learning aspects of AV** software [28], [29], [51], [52], [53], [54], [55], [56]. Rather than focus on these aspects, SCENORITA targets the planning component of AV software. Previous work has shown that the most bug-ridden component of production-grade, open-source AV software systems is the planning component as opposed to the components responsible for or utilizing vision or machine-learning capabilities of AVs [57].

Reproducing tests from real crashes. Crashes are recreated by replaying the sensory data collected during physical-world crashes in [58]. Similarly, *AC3R* [31] generates driving simulations that reproduce car crashes from police reports using natural language processing (NLP). However, *AC3R* requires manual collection of police reports and inherits the accuracy limitations of the underlying NLP used to extract information from police reports.

Search-based procedural road generation. *AsFault* [25] uses procedural content generation and search-based testing to automatically create challenging virtual scenarios for AV software. Similarly, tools published as part of the Search-Based Software Testing Challenge (SBST) [59], [60] generate challenging road networks for virtual testing of an automated lane-keeping system such as GABezier [61], Frenetic [62], and Deeper [63]. However, none of these tools take into account the behavior of other obstacles when testing for safety violations in AVs. Other tools in the SBST Challenge derive tests for Java such as EvoSuite [64] and Kex [65]. None of the latter tools are targeted to generate tests for autonomous vehicles.

Next, we discuss **interesting but orthogonal research problems** such as Li et al. [66] which discussed the original idea to consider the safety and comfort of autonomous vehicles. However, this work does not formalize or encode safety and comfort—when evolving driving scenarios—to produce tests that expose the autonomous vehicle to safety and comfort violations. This work, instead, aims to provide a quantitative way to measure the safety and comfort of autonomous driving in a test. Another major distinction between scenoRITA and Li et al.'s approach is that the latter is not a fully automatic test generation framework. This approach requires a human expert to vaguely define "test tasks" and perform qualitative judgments before the simulation-based system can make more precise task definitions and generate more tests. The human expert then provides feedback to the simulation system to validate test results.

Another orthogonally related work is by Calò et al. [26] which proposed two search-based approaches for finding *avoidable* collisions. They define comfort and speed as weights to rank short-term paths; however, they do not formalize comfort and speed in the fitness function, nor do they evaluate them. The main focus of our paper is not to introduce a sophisticated approach to evaluate violations' "avoidability" but to find as many violations as possible. Nonetheless, we still ensure that the generated tests are valid using time-, speed-, and location-related thresholds to determine "avoidable" violations.

As opposed to finding safety and comfort violations, Luo et al. propose a framework (EMOOD) [67] that evolves tests to identify combinations of requirements violations. In other words, EMOOD aims to find different requirements violation patterns for two reasons: (i) different combinations of requirements violations can expose different types of failures. For example, the type of failure in which the autonomous vehicle collides while running a red light is different from the one in which the autonomous vehicle collides while violating lane keeping, as the different combinations of requirements violations may provide different insights about the cause of the collisions; (ii) satisfying all requirements may not be possible for an Autonomous Driving System (ADS) in practice, as unexpected events may happen in highly open and dynamic environments. In response to these unexpected events, the control software of the ADS has to make trade-offs among requirements, likely resulting in one or more requirements being violated.

Table I compares SCENORITA with the state-of-the-art techniques that are most closely related. *Scenario Types* in Table I refer to the number of different scenarios supported by each approach. A scenario type is determined by (i) a specific part of a map (e.g., intersection X in map Y), (ii) the number of obstacles in that scenario, (iii) the movement path/routing of obstacles and the ego car (e.g., obstacle O starts at point A and ends at point B), (iv) the allowed maneuvers of obstacles and the ego car (e.g., obstacle O only turns right at an intersection). For example, [68] generates tests for **one scenario type** only, which consists of **one pedestrian** crossing the **same street** in a given map, with fixed trajectories for both the ego car and pedestrian, throughout the test generation process. Another approach [69] generates driving scenarios where obstacles only perform a right or left lane change and no other maneuvers are

TABLE I
COMPARING SCENORITA WITH THE MOST-SIMILAR RELATED WORK

	SCENORITA	AV-FUZZER [24]	AutoFuzz [30]
Main Objective	Generate driving scenarios that expose the ego car to **3** types of **safety**-critical and **2** types of **motion** sickness-inducing scenarios	Find **collision** violations of autonomous vehicles in the presence of an evolving traffic environment	A grammar-based fuzzing technique for finding **collision** and **off-road violations** in AV controllers
Supports High Definition (HD) Maps	Yes	Yes	Yes
Representations (gene)	Fully Mutable	Partially Mutable	Partially Mutable
Scenario Types	Unlimited	Limited (1 scenario: the ego car and obstacles travel along a lane)	Limited (2 scenarios: (1) going across a junction, (2) turning left at a junction)
Scenario Configuration	Automated	Manual	Manual
Supported Maneuvers	Supports **any combination** of the following maneuvers: Lane Follow, Lane Change, Left Turn, Right Turn, Cross Intersection, U-Turn, Lane Merge, Acceleration, Deceleration.	Supports **one** maneuver at a time from 4 possible maneuvers: Acceleration, Deceleration, Lane Follow, and Lane Change.	Supports **one** maneuver at a time from 6 possible maneuvers: Acceleration, Deceleration, Turn Left, Turn Right, Lane Follow, and Lane Change.
Ego Car Routing	Flexible	Fixed	Fixed
No. Obstacles Per Scenario	10–30	2	1, 2 or 4 (depending on the manual specification)
Collision Detection	Supported	Supported	Supported
Speeding Detection	Supported	Not Supported	Not Supported
Unsafe Lane Change	Supported	Not Supported	Not Supported
Fast Acceleration	Supported	Not Supported	Not Supported
Hard Braking	Supported	Not Supported	Not Supported
No. Manually-Specified Variables Per Scenario Type	0	12	10 + (20 Per Obstacle)

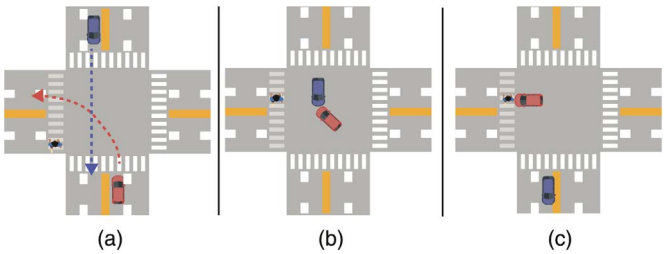


Fig. 1. An illustration of one of five supported driving scenarios in AutoFuzz. (a) The ego car (in red) starts at a fixed location and then turns left at a signalized junction, while another vehicle (in blue) crosses the intersection from the other side and a pedestrian crosses the street. (b) The ego car turns left and collides with an incoming car (in blue). (c) The ego car turns left and collides with a pedestrian crossing the street.

supported. Similarly, [70] generates 4 scenario types where: the ego car only turns left at an intersection in one scenario, only drives straight in two others, and only performs a lane change in the last scenario. The approaches depicted in Table I and similarly [67], [68], [69], [70], [71], [72], [73] fail to find diverse violations due to the limited number and lack of diversity in generated scenarios (the maximum number of scenario types supported by any of these tools is 7 scenario types).

Scenario Configuration in Table I refers to the means used to set up a *Scenario Type*. For example, to set up a *Scenario Type* similar to Fig. 1(a) in AutoFuzz [30], one must manually set (i) start and end location of the ego car in a specific map, (ii) allowed maneuver of the ego car (e.g., in Fig. 1(a), the ego car always turns left), (iii) the number and type of obstacles (e.g., in Fig. 1(a), there's one car and one pedestrian), (iv) start and end locations for every possible obstacle in that scenario (e.g., in Fig. 1(a), one must specify the routing info of the blue car and the pedestrian). Any slight difference in one scenario type in AutoFuzz, requires manually modifying its configuration. We believe that the reason for the limited number of scenario types and limited maneuvers in related work is due to the fact that these tools require a manual setup for driving scenarios. For example, to generate a scenario other than the ones specified by the authors of these tools, a typical developer has to go through (1) using an HD map viewer to visualize the map, (2) selecting a point on the map to place the ego vehicle, (3) finding the coordinate of the selected point, and (4) calculating the vehicle's heading. After correctly completing these 4 steps, a developer will have a single waypoint that specifies a coordinate and heading of the initial location of the ego car. The same 4-step process needs to be repeated for specifying the ego car's final location, and the locations of all other road traffic participants. All of these steps are difficult to accomplish since HD maps are difficult for humans to read and cannot be visualized without a dedicated tool.

Ego Car Routing in Table I indicates whether an approach supports *flexible* ego car routing (i.e., an ego car can traverse different paths in a map, covering as many roads of the map as possible) or *fixed* routing (i.e., ego car traverses the same path over and over again), hence, must be defined manually.

The need to manually set up these scenarios can be attributed to the gene representation used in related work. These tools avoid applying search operators on an obstacle attribute level (unless it's the speed attribute) due to the tools' inability to

generate obstacles with valid new trajectories. For example, applying a crossover between two obstacles can result in an offspring with a new movement path (trajectory); hence, there needs to be a mechanism that ensures that the newly generated obstacle has a valid trajectory (i.e., travels in the direction of traffic, with close proximity to the ego car but still allows the ego car ample amount of time to react to any potential violations, not placed in the middle of intersections, etc.). The challenging aspect in generating complex and diverse scenarios is to be able to evolve obstacles fully regardless of the initial setup of such obstacles in a scenario. Related work evolves obstacles in tests by either: (i) transferring obstacles across scenarios without altering their states or properties, or (ii) mutating the speed of such obstacles or moving them to neighboring lanes. As a result, these works opt for using fixed, manually-setup scenarios—throughout test generation—with obstacles and the ego car having fixed trajectories and limited maneuvers.

SCENORITA aims to address the fundamental research challenge of burdensome human efforts in ADS testing by analyzing the HD map so that waypoints can be automatically generated. After automatically identifying the initial and final location of the obstacle, it uses algorithms used by the ADS' routing module to compute the lanes the obstacle needs to travel on and uses robotics algorithms (e.g., the Stanley Controller, PID speed control) to execute the maneuvers. As a result, any obstacle can execute complex and smooth trajectories without any human effort. By not involving any human effort, scenarios that SCENORITA generates cover a significantly larger range of road structures and traffic control devices (i.e., traffic lights, and stop signs), which we will elaborate on in Section V-C.

Moreover, the approaches described in this section, only find collision violations [24], [26], [30], [68], [69], [70], [71], [72] or test ADS' lane-keeping capability [25], [61], [62], [63], [73]. None of them evolve tests for a combined set of safety violations (collision, speed, and unsafe lane change), and none of them report comfort violations.

III. Specification of the State Space

To aid in the generation of effective and valid scenarios, we present a formal specification of the state space in the form of driving scenarios. SCENORITA uses this formal specification of the state space, along with a genetic algorithm, to generate scenarios that maximize the possibility of the ego car (i.e., AV) either violating safety or causing rider discomfort.

Definition 1: A Scenario $\mathcal{S}_c$ is a tuple $\langle t, E, \mathbb{O}, \mathbb{L}\rangle$ where:

- t is a finite number that represents the maximum duration of $\mathcal{S}_c$.
- E is the only ego car (i.e., the autonomous driving car) in $\mathcal{S}_c$.
- $\mathbb{O}$ is a finite, non-empty set of n obstacles (i.e. non-player characters). A *single* obstacle is represented as O_k where $\mathbb{O} = \{O_k : 1 \leq k \leq n\}$.
- $\mathbb{L}$ is a finite, non-empty set of lanes, where E and $\mathbb{O}$ reside/travel.

Definition 2: An ego car E is a tuple $\langle Z_E, \mathbb{P}_E, \mathbb{S}_E, \mathbb{A}_E, \mathbb{C}_E, \mathcal{PD}_E\rangle$ where:

- $Z_E = \langle wid, len, hgt\rangle$ represents the width wid, length len, and height hgt of the ego car E.
- $\mathbb{P}_E$ is a finite, non-empty set representing the ego car's positions during time instants of $\mathcal{S}_c$. The position of E at timestamp j is represented as p_j^E where $\mathbb{P}_E = \{p_j^E : 1 \leq j \leq t\}$.
- $\mathbb{S}_E$ is a finite, non-empty set representing the ego car's speed during time instants of $\mathcal{S}_c$. The speed of E at timestamp j is represented as s_j^E where $\mathbb{S}_E = \{s_j^E : 1 \leq j \leq t\}$.
- $\mathbb{A}_E$ is a finite, non-empty set representing the ego car's acceleration at time instants of $\mathcal{S}_c$. The acceleration of E at timestamp j is represented as a_j^E where $\mathbb{A}_E = \{a_j^E : 1 \leq j \leq t\}$.
- $\mathbb{C}_E$ is a finite, non-empty set of durations an ego car spends driving at the boundary of two lanes at the same time. When an ego car changes lanes, it drives on the markings between two lanes for a period of time c, before it completely switches to the target lane. The duration E spends driving on the markings at timestamp j is represented as c_j^E where $\mathbb{C}_E = \{c_j^E : 1 \leq j \leq t\}$.
- $\mathcal{PD}_E = \{pd_j^E : 1 \leq j \leq t\}$ is a finite, non-empty set of planning decisions at time instants of $\mathcal{S}_c$. A planning decision is a pair $\langle t_{pd}^E, \mathbb{P}_\mathcal{P}\rangle$ produced by the ego car's planning module where:
  - t_{pd}^E is the timestamp at which the planning decision $\mathcal{PD}_E$ was made.
  - $\mathbb{P}_\mathcal{P} = \langle (x_1, y_1, t_1), (x_2, y_2, t_2), ..., (x_n, y_n, t_n)\rangle$ is a finite sequence representing the ego car's planned positions, relative to timestamp t_{pd}^E, where the ego car plans to reach position (x_j, y_j) at timestamp $t_{pd}^E + t_j$ and $1 \leq j \leq n$.

Definition 3: A single obstacle O_k in $\mathcal{S}_c$ is a tuple $\langle ID_{O_k}, T_{O_k}, Z_{O_k}, M_{O_k}, \mathbb{P}_{O_k}, \mathbb{S}_{O_k}\rangle$ where:

- ID_{O_k} represents a unique identification number associated with O_k.
- T_{O_k} represents the type of an obstacle. Examples of obstacle types are: `VEHICLE`, `PEDESTRIAN`, and `BICYCLE`.
- $Z_{O_k} = \langle wid, len, hgt\rangle$ represents the width wid, length len, and height hgt of obstacle O_k.
- M_{O_k} represents the mobility of an obstacle (e.g., static or mobile).
- $\mathbb{P}_{O_k}$ is a finite, non-empty set representing the positions of O_k at time instants of $\mathcal{S}_c$. The position of O_k at timestamp j is represented as $p_j^{O_k}$ where $\mathbb{P}_{O_k} = \{p_j^{O_k} : 1 \leq j \leq t\}$.
- $\mathbb{S}_{O_k}$ is a finite, non-empty set representing the speed of O_k at time instants of $\mathcal{S}_c$. The speed of O_k at timestamp j is represented as $s_j^{O_k}$ where $\mathbb{S}_{O_k} = \{s_j^{O_k} : 1 \leq j \leq t\}$.

Definition 4: A single lane l in $\mathbb{L}$ is a tuple $\langle \mathbb{S}_l, \mathbb{P}_l\rangle$ where:

- $\mathbb{S}_l$ is a finite, non-empty set representing the speed limit imposed by l. The speed limit of l, which the ego car is traversing at timestamp j, is represented as s_j^l where $\mathbb{S}_l = \{s_j^l : 1 \leq j \leq t\}$.
- $\mathbb{P}_l$ is a finite, non-empty set representing the position of the closest lane boundary to the ego car. The position of

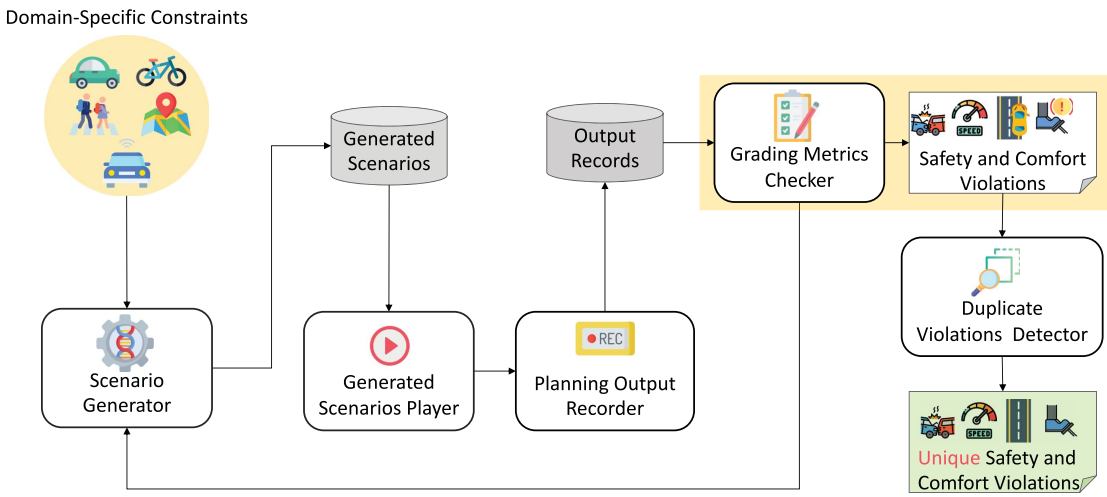


Fig. 2. An Overview of SCENORITA.

l's boundary, which the ego car is traversing at timestamp j, is represented as p_j^l where $\mathbb{P}_l = \{p_j^l : 1 \leq j \leq t\}$.

Definition 5: We define a violation $v \in \mathbb{V} = \{collision, speed, unsafeChange, fastAccl, hardBrake\}$. We elaborate on the oracles corresponding to each of these violations in Section IV-E.

IV. SCENORITA

Fig. 2 shows the overall workflow of SCENORITA. Our main goal is to create valid and effective driving scenarios that expose AV software to unique safety and comfort violations. SCENORITA achieves this goal as follows: (1) it takes as an input a set of *domain-specific constraints*, which dictates what constitutes a *valid* driving scenario (e.g., obstacles should be moving in the direction of traffic in the lane and have valid obstacle identifiers); (2) The *Scenario Generator* uses a genetic algorithm to produce genetic representations of driving scenarios with randomly generated but valid obstacles, following the *domain-specific constraints*. The genetic algorithm evolves the driving scenarios with the aim of finding scenarios with safety and comfort violations; (3) *Generated Scenarios Player* converts the genetic representation of scenarios (*Generated Scenarios*), from the previous step, into inputs to the ADS and executes a scenario. The planning output of the ADS under test will be produced by the ADS's planning module and recorded by *Planning Output Recorder*; (4) The planning output is then evaluated by *Grading Metrics Checker* for *safety and comfort violations*; (5) When the evolution process terminates, the *Duplicate Violations Detector* inspects the violations produced by *Grading Metrics Checker* to eliminate any duplicate violations and produces a set of *unique safety and comfort violations*. In the remainder of this section, we discuss each of these elements of SCENORITA in more detail.

A. Domain-Specific Constraints

Table II specifies the list of constraints that *Scenario Generator* should follow to ensure that the generated driving scenarios are *valid*. In this work, we define *valid scenarios* as those which contain obstacles that are (1) moving in the direction of traffic in the lane; (2) having start and end points contained within the boundaries of a fixed-size map; and (3) having dimensions (i.e. width, height, and length) and speed that account for the obstacle type (vehicle, pedestrian, or bike). For example, the speed of a pedestrian should not exceed the average walking/running speed of a human.

When generating the initial and final position of the ego car and obstacles, *Scenario Generator* must ensure that these points are (i) within the boundaries of a fixed-size map, and (ii) have a valid path allowing the ego car and obstacles to move in the direction of traffic.

Unlike prior work, we consider a wide range of obstacle-related attributes including type, size, speed, and mobility. An obstacle can be a `VEHICLE`, `BICYCLE`, or a `PEDESTRIAN`, and the type of an obstacle dictates the minimum and maximum allowed values of its size and speed. An obstacle is represented as a polygon, hence, its size is expressed in terms of the width, height, and length (i.e., Z_{O_k}) of the polygon. The ego car E is similarly represented as a polygon based on Z_E.

To determine the maximum and minimum dimensions of a pedestrian, we followed the most recent report published by the National Center for Health Statistics (NCHS) [43], which provides the most recent anthropometric reference data for children and adults in the United States. The height of a pedestrian ranges from 0.97 m (average height of a child) to 1.87 m (average height of an adult aged 20+). The width (shoulder width) ranges from 0.24 m to 0.67 m, while the length ranges from 0.2 m to 0.45 m. The speed of a pedestrian can range from 4.5 km/h (average walking speed) to 10.5 km/h (average running speed) [74].

To determine the maximum and minimum dimensions and speed for both bicycles and vehicles, we followed the size and speed regulations imposed by the Federal Highway Administration and the US Department of Transportation [44], [45]. The speed of a bicycle can range from 6 km/h to 30 km/h, while the speed of a vehicle can range from 8 km/h (e.g., parking lots) to 110 km/h (e.g., highways).

TABLE II
A LIST OF *DOMAIN-SPECIFIC CONSTRAINTS* THAT *SCENARIO GENERATOR* ADHERES TO WHEN CREATING DRIVING SCENARIOS

Scenario Attr.	Sub-Attr.	Description	Constraints
Ego Car	**Initial Position**	The start position of the ego car in the map	Initial Position should be within the map boundaries
	Final Position	The destination position of the ego car in the map	Final Position should be within the map boundaries, and there should be a valid path between the ego car's Initial and Final Position
Obstacles	**ID**	A unique identification number associated with each obstacle in a Scenario	Obstacles in a single Scenario have unique IDs
	Initial Position	The start position of an obstacle in the map	Initial Position should be within the map boundaries
	Final Position	The destination position of an obstacle in the map	Final Position should be within the map boundaries, and there should be a valid path between the obstacle's Initial and Final Position
	Type	The type of an obstacle	An Obstacle can be one of the following values: (`VEHICLE`, `BICYCLE`, `PEDESTRIAN`)
	Speed	Maximum speed of an obstacle, measured in km/h. The obstacle type dictates the valid minimum and maximum speed of an obstacle:	
		`VEHICLE`	The speed of a vehicle can range from 8 km/h (e.g., parking lots) to 110 km/h (e.g., highways)
		`BICYCLE`	The speed of a bicycle can range from 6 km/h to 30 km/h
		`PEDESTRIAN`	The speed of a pedestrian can range from 4.5 km/h (average walking speed) to 10.5 km/h (average running speed)
	Width	Width of an obstacle, measured in meters. The obstacle type dictates the valid minimum and maximum width of an obstacle:	
		`VEHICLE`	[1.5 - 2.5] in meters
		`BICYCLE`	[0.5 - 1] in meters
		`PEDESTRIAN`	[0.24 - 0.67] in meters
	Length	Length of an obstacle, measured in meters. The obstacle type dictates the valid minimum and maximum length of an obstacle:	
		`VEHICLE`	[4 - 14.5] in meters
		`BICYCLE`	[1 - 2.5] in meters
		`PEDESTRIAN`	[0.2 - 0.45] in meters
	Height	Height of an obstacle, measured in meters. The obstacle type dictates the valid minimum and maximum height of an obstacle:	
		`VEHICLE`	[1.5 - 4.7] in meters
		`BICYCLE`	[1 - 2.5] in meters
		`PEDESTRIAN`	[0.97 - 1.87] in meters
	Motion	The motion of an obstacle	An Obstacle can either be: *static* or *mobile*

B. Scenario Generator

Our overarching goal is to create valid and effective driving scenarios that expose AV software to safety and comfort violations. The *Scenario Generator* takes as input a set of *domain-specific constraints* (Section IV-A) and uses a genetic algorithm to maximize a defined set of fitness functions (representing safety and comfort violations) to guide the search for problematic scenarios. The genetic algorithm is initialized with a starting population of obstacles; these obstacles form groups, and each group of obstacles represents road-traffic participants for a single scenario. To evaluate the fitness of obstacles, scenario representations are transformed into driving simulations, in which navigation plans are generated based on the origin and destination of the ego car. Additionally, driving trajectories/plans are computed for the ego car based on the scenario set-up (e.g., number of obstacles, state of the obstacles, ego car start and target position, etc.). For example, the ego car's start position should be outside of any intersection and with sufficient distance away from other obstacles. During the simulation, the driving decisions of the ego car (e.g., driving maneuvers, stop/yield decisions, acceleration) are recorded by *Planning Output Recorder* at regular intervals to identify safety and comfort violations. A set of values such as the distance between the ego car and other obstacles, the distance between the ego car and lane boundaries, the speed of the ego car, and the acceleration and deceleration of the ego car are used to compute the fitness of obstacles (Section IV.B.2). These values guide the genetic algorithm when evolving obstacles by recombining and mutating their attributes (Section IV.B.3). The algorithm continues to execute and evolve obstacles until a user-defined ending condition is met, at that point, SCENORITA returns the final test suite and stops.

1) Representation: Fig. 3(a) illustrates the genetic representation of an individual generated by SCENORITA. Individuals from the population are grouped into sub-populations, i.e., demes, that, in turn, represent scenarios. Each deme has a different ego car and a set of obstacles from other demes, and obstacles evolve within a deme independently from other demes.

An individual is represented as a vector, where each index in the vector represents a gene. The number of input genes is fixed, where the 9 genes correspond to the following 9 attributes

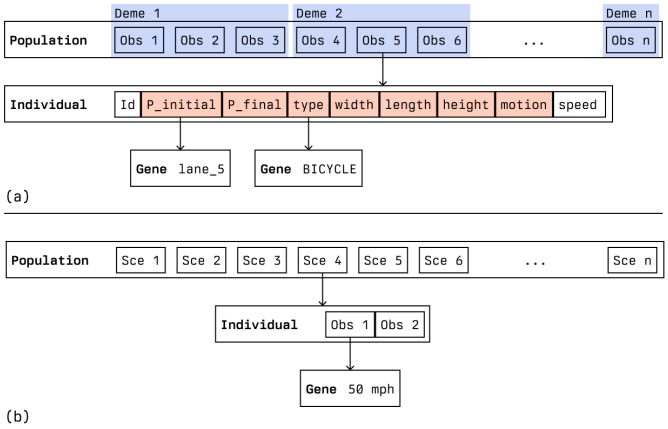


Fig. 3. Genetic representation of fully-mutable tests in SCENORITA (a), and partially mutable tests generated by state-of-the-art approaches (b).

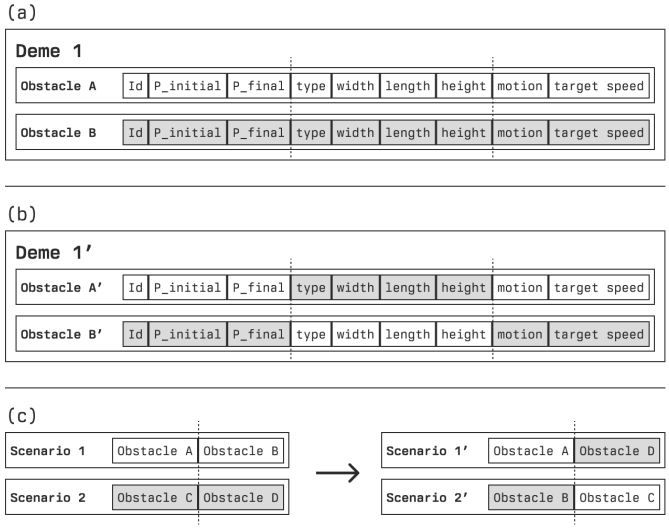


Fig. 4. (a) Two individuals (obstacles) *before* a crossover, (b) the same individuals (obstacles) *after* a crossover for SCENORITA, and (c) how crossover is applied in prior work [24].

of a *single* obstacle O_k: ID_{O_k}; the initial position $p_1^{O_k}$ of O_k; the final position $p_t^{O_k}$ of O_k at the final timestamp t; length $Z_{O_k}.len$; width $Z_{O_k}.wid$; and height $Z_{O_k}.hgt$; initial speed $s_i^{O_k}$ at timestamp j; type T_{O_k}; and mobility M_{O_k}. Each gene value can change (e.g., when initialized or mutated), but it still has to adhere to the valid ranges defined in Section IV-A.

One of the key insights of our approach is enabling full mutability of obstacles and, thus, greater diversity by representing obstacles as individuals instead of genes. To that end, we use a non-traditional means of relating obstacles to scenarios by introducing intermediate genetic elements (i.e., demes) that represent driving scenarios. Fig. 3(b) shows the genetic representation of tests that other meta-heuristic approaches utilize, where individuals represent scenarios and obstacles represent genes, significantly limiting the ability to modify an obstacle.

Representing *obstacles* as *individuals* allows SCENORITA to alter obstacles' attributes and states when applying search operators, hence, allowing obstacles to be *fully mutable* throughout the test generation process. Fig. 4(b) demonstrates SCENORITA's application of a crossover operator on two individuals (i.e., obstacles) compared to how related work recombines their individuals (Fig. 4c). Previous approaches [24], [25], [26], represent *obstacles* as *genes*, resulting in obstacles being mostly the same across all scenarios generated. For example, obstacle attributes in AV-FUZZER [24] cannot be altered, resulting in scenarios in which the same obstacles always start from the same location. Prior approaches also have a limited selection of obstacles because they have to choose from what the simulator supports (e.g., sedans, SUVs, trucks, and a school bus), and a user may not change the dimensions of any of the obstacle types. To overcome this limitation, SCENORITA includes obstacle dimensions in the representation and, as a result, can generate obstacles with more diverse dimensions and obstacle properties.

By incorporating more fine-grained attributes (e.g., location) into the representation of obstacles, SCENORITA must address the challenge of ensuring the creation of valid obstacle trajectories. To address this challenge, we leverage the insight that path-finding algorithms used by the routing module of the ADS can be repurposed to find valid paths for the obstacles. SCENORITA also uses those algorithms to check whether there is a valid path between the start and end location of an obstacle, and further corrects invalid start and end locations that may have resulted from the mutation and crossover operators. This process allows SCENORITA to automatically generate valid obstacle paths instead of requiring human effort to specify each path and is shown to be more effective in finding violations and more diverse in terms of covering different parts of the map. We will further elaborate on this in Section V-B and Section V-C.

2) Fitness Evaluation: In each generation, individuals are assessed for their fitness, with respect to the search objective, to be selected to pass on their genes. SCENORITA determines the fitness of an individual by evaluating how close they are in terms of causing safety or comfort violations. This is measured by calculating the fitness of an individual i using a function $f_v(i)$ with respect to the safety and comfort violation v. Recall from Section I that SCENORITA considers 5 safety and comfort violations based on the grading metrics defined by Apollo's developers [35] and, thus, represents violation constructs and thresholds used by professional AV developers. Three of these metrics assess driving scenarios for traffic and road safety (collision detection, speeding detection, and unsafe lane change), while the remaining two metrics assess a rider's comfort (fast acceleration and hard braking).

The fitness of an individual i is determined as follows:

$$F(i) = \Big(f_{collision}(i), f_{speed}(i), f_{unsafeChange}(i), f_{fastAccl}(i), f_{hardBrake}(i)\Big) \quad (1)$$

Recall that $v \in \mathbb{V} = \{collision, speed, unsafeChange, fastAccl, hardBrake\}$ (Definition 5), hence, $F(i)$ aims to maximize the number of violations. In the remainder of this section, we define $f_v(i)$ in more detail.

Collision Detection. In the context of collision detection, effective tests are those which cause the ego car to collide with other obstacles. Therefore, SCENORITA uses as a fitness

function $f_{collision}$ (Equation 2), which rewards obstacles that move as close as possible to the ego car. Given a simulated scenario with a maximum duration of t, a set of positions $\mathbb{P}_E$ for the ego car E, and a set of individuals (i.e., obstacles), the position of each individual i in the scenario is defined as $\mathbb{P}_i = (p_1^i, p_2^i, ..., p_t^i)$. From these, we define $f_{collision}$ as:

$$f_{collision}(i) = \min\{D_c(p_j^E, p_j^i) \mid 1 \leq j \leq t \wedge p_j^E \in \mathbb{P}_E \wedge p_j^i \in \mathbb{P}_i\} \quad (2)$$

$D_c(p^E, p^i)$ is the shortest distance between the position of individual i and the ego car at a given time. $f_{collision}$ captures the intuition that obstacles that move closer to the ego car (i.e., have a minimum distance between the ego car and the obstacle) are more likely to lead to a collision.

Speeding Detection. We use a fitness function f_{speed} (Equation 3), which rewards obstacles that cause the ego car to exceed the speed limit or the current lane. Given a simulated scenario with a maximum duration t, the speed profile $\mathbb{S}_E$ of the ego car E, and a set of speed limits imposed by lanes of which the ego car traversed $\mathbb{S}_L$, we define f_{speed} as:

$$f_{speed}(i) = \min\{D_s(s_j^l, s_j^E) \mid 1 \leq j \leq t \wedge s_j^E \in \mathbb{S}_E \wedge s_j^l \in \mathbb{S}_L\} \quad (3)$$

s_j^E and s_j^l represent the ego car speed and the speed limit of the lane in which the ego car is traveling at timestamp j, respectively. Furthermore, $D_s(s^l, s^E)$ is the difference between the speed limit imposed by a given lane and the current speed of the ego car. f_{speed} captures the intuition that as the ego car approaches the speed limit of a given lane it is more likely to result in speed violations.

Unsafe Lane Change. A lane change is defined as a driving maneuver that moves a vehicle from one lane to another, where both lanes have the same direction of travel. We primarily focus on the *duration* the ego car spends traveling at the boundary of two lanes while changing lanes. We define a *safe* lane-change duration as δ_{safe}. We define a fitness function $f_{unsafeChange}$ (Equation 4), which rewards obstacles that cause the ego car to spend more than δ_{safe} driving at the boundary of two lanes. Given a simulated scenario with lane-change durations $\mathbb{C}_E$ for ego car E, and a safe lane-change duration δ_{safe}, we define $f_{unsafeChange}$ as:

$$f_{unsafeChange}(i) = \max\{c_j^E \mid 1 \leq j \leq t\} \quad (4)$$

$c_j^E \in \mathbb{C}_E$ represents the duration an ego car spends driving between two lanes. $f_{unsafeChange}$ captures the intuition that obstacles causing the ego car to spend longer periods of time driving on lane boundaries are more likely to result in an unsafe lane change violation.

Fast Acceleration. We use a fitness function $f_{fastAccl}$ (Equation 5), which rewards obstacles that cause the ego car to accelerate too fast, potentially inducing motion sickness. Given a simulated scenario with a maximum duration t and the acceleration profile $\mathbb{A}_E$ for the ego car E, we define $f_{fastAccl}$ as:

$$f_{fastAccl}(i) = \max\{a_j^E \mid 1 \leq j \leq t \wedge a_j^E \in \mathbb{A}_E\} \quad (5)$$

$a_j^E \in \mathbb{A}_E$ represents the acceleration of the ego car E at timestamp j. $f_{fastAccl}$ aims to maximize the acceleration of E to induce a motion sickness violation.

Hard Braking. We use a fitness function $f_{hardBrake}$ (Equation 6), which rewards obstacles that cause the ego car to brake too hard (i.e., brake suddenly in a manner that induces motion sickness). Given a simulated scenario with a maximum duration t, the acceleration profile for the ego car defined as $\mathbb{A}_E$, we define $f_{hardBrake}$ as:

$$f_{hardBrake}(i) = \min\{a_j^E \mid 1 \leq j \leq t \wedge a_j^E \in \mathbb{A}_E\} \quad (6)$$

$a_j^E \in \mathbb{A}_E$ represents the deceleration of the ego car. Similar to the previous fitness function, $f_{hardBrake}$ aims to maximize the deceleration of E to induce a hard-braking violation.

3) Search Operators: SCENORITA evolves driving scenarios by applying search operators, which mutate and recombine the scenario attributes according to certain probabilities. In this section, we provide a detailed explanation of these search operators.

Selection. SCENORITA uses the **N**on-dominated **S**orting **G**enetic **A**lgorithm selection (NSGA-II [75]) for breeding the next generation. NSGA-II is an effective algorithm used for solving multi-objective optimization problems (i.e., problems with multiple conflicting fitness functions) and further aims to maintain the diversity of individuals.

NSGA-II starts by sorting a set of individuals based on a *non-dominated* order. In a multi-objective problem, an individual i_1 is said to *dominate* another individual i_2 if (1) i_1 is no worse than i_2 for **all** objective functions (e.g., collision detection, speeding detection, etc.), and (2) i_1 is strictly better than i_2 in at least one objective. Once the non-dominated sort is complete, a *crowding distance* is assigned to every individual in a given scenario. A *crowding distance* measures how close individuals are to each other; a large average crowding distance will result in better diversity in the population. Once the crowding distance is assigned, parent individuals are selected to produce offspring based on the fitness and crowding distance; an individual is selected if its order rank is less than the other, or if the crowding distance is greater than the other. Only the best N individuals are selected, where N is the population size.

The intuition behind using NSGA-II selection is threefold: (1) it uses an elitist principle, i.e., the most elite individuals in a scenario are given the opportunity to be reproduced so their genes can be passed on to the next generation; (2) it uses an explicit diversity-preserving mechanism (i.e., crowding distance), which maintains the diversity of driving scenarios in SCENORITA; and (3) it emphasizes the non-dominated solutions.

Crossover. This operator selects two individuals (obstacles) from a given deme (scenario) and creates superior offspring by mixing their parents' genetic makeup. SCENORITA uses a two-point crossover strategy, where two crossover points are picked randomly from the mating individuals (i.e., parent obstacles) and the genes between the two points are swapped. Fig. 4 illustrates the application of the two-point crossover operator on two sample individuals; the two individuals are modified in place and both keep their original length. We opt for the two-point

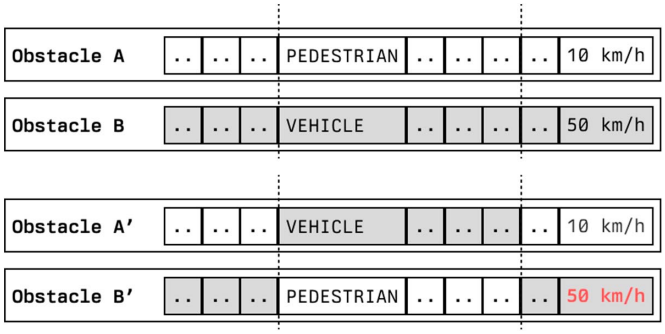


Fig. 5. An example of a crossover that produced individuals with invalid attributes (highlighted in red).

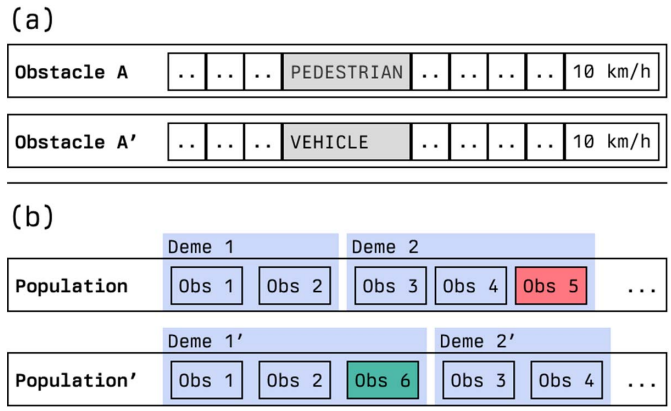


Fig. 6. (a) Mutating a gene in a *single* individual, (b) mutating a deme by adding an individual and mutating a deme by removing an individual.

crossover strategy since it maintains the length of individuals, in addition to increasing the extent of disruption in their original values. SCENORITA applies its two-point crossover between obstacles in the same scenario (i.e., the same deme) to create new obstacles; unlike prior work, where crossover is applied at the scenario level by swapping obstacles across different scenarios, making these previous approaches unable to create new obstacles.

Crossover may produce invalid scenario configurations. For example, after crossover of a vehicle's attributes with those of a pedestrian (Fig. 5), some obstacle attributes produced in the offspring may violate the speed and size constraints in Section IV-A, such as having a pedestrian's speed changed from **10** km/h (a valid running speed for a pedestrian) to **50** km/h (an unrealistic speed for a pedestrian). When such a case is detected, SCENORITA replaces the violated obstacle attributes with randomly generated values that fall within the valid ranges described in Section IV-A.

Mutation. SCENORITA applies two forms of mutation operators: (1) it mutates a gene (attribute) of an individual (obstacle); and (2) it applies the mutation operator to a deme (i.e., a scenario). The first type of mutation randomly replaces attributes of obstacles with new ones, where the newly generated values follow the constraints defined in Section IV-A. For example, the mutation operator can change the speed of a vehicle from **35** km/h to **50** km/h. This type of mutation does not change the number of obstacles in a single scenario. The second type of mutation aims to diversify the *number* of obstacles in a given scenario $\mathcal{S}_c$ by (1) adding a randomly generated obstacle to $\mathcal{S}_c$ or (2) removing a random obstacle from $\mathcal{S}_c$.

Fig. 6(a) shows the mutation of an individual's type gene, changing from pedestrian to vehicle; Fig. 6(b) shows the mutation of demes, by adding an individual or removing an individual.

C. Generated Scenarios Player

Converting the genotype representation of tests to driving simulations is the key task of *Generated Scenarios Player*. While the generated scenario representation specifies the start and the end location of each obstacle, it does not specify *how* it should move during the simulation.

Generated Scenarios Player is responsible for modeling obstacle movements and publishing ground truth obstacle positions to the ADS at every timestamp j, and position $p_j^{O_k}$ of obstacle O_k is computed based on the modeling algorithm. SCENORITA leverages the insight that state-of-the-art robotics algorithms, i.e., a steering control algorithm (e.g., Stanley Controller [76]) and a speed control algorithm (e.g., PID speed control [77]), can be used in combination to simulate the movement of the obstacles. A steering control algorithm aims to ensure the obstacle follows lanes on the map to reach its destination defined by *Scenario Generator*. Recall from Section IV.B.1, *Scenario Generator* only specifies the start and end location of an obstacle. *Generated Scenario Player* then simulates the movement of the obstacle along the shortest path on the map from the start to the end location, and computes the obstacle's position $\mathbb{P}_{O_k}$ along with its speed $\mathbb{S}_{O_k}$. The final sequence is then transformed into real-time perception information, and in turn, used as input for the planning module of the ADS during the scenario.

Once the positions and speeds of obstacles are computed, *Generated Scenarios Player* starts the driving simulator, where the planning module computes the driving trajectory taking into consideration various aspects of the vehicle and elements of its environment (e.g., distance to lane center, smoothness of the trajectory).

D. Planning Output Recorder

The behavior of the ego car in the driving simulation (Section IV-C) is recorded by the *Planning Output Recorder*, which stores the ego car's driving behaviors in a record file. The recorded output enables *Grading Metrics Checker* to identify and report the occurrences of *safety and comfort violations*. These violations are checked when the driving scenario ends; hence, it does not halt the driving simulation after observing the first violation; instead, tests continue until the end of the scenario. This approach balances the cost of running expensive simulations with the benefit of collecting as many violations as possible. Note that we do not count any violations occurring after the first collision in a generated scenario.

SCENORITA uses the output records for reporting violations, and evaluating the fitness of tests which guides the evolution process in Section IV-B. Additionally, stored records enable us

to replay scenarios with reported violations after SCENORITA ends; this allows us to verify the correctness of generated tests, and to closely analyze them along with their underlying causes.

E. Grading Metrics Checker

Previous work [24], [25], [26], [28], [29], [30], [31] considers a limited number of test oracles, mainly consisting of one test oracle per work (either collision detection or lane keeping). The limited use of test oracles found in such techniques ignores important safety and comfort issues (e.g., driving between lanes for too long or causing motion sickness) and provides significantly less insight into the testing of industry-grade AVs. Unlike previous work, we consider 5 test oracles based on grading metrics defined by Apollo's developers [35]. These grading metrics test different aspects of AVs, ranging from traffic and road safety to a rider's comfort. In the remainder of this section, we describe each grading metric in detail along with the definition of its corresponding test oracle.

The **Collision Detection** oracle checks if the ego car reaches its final destination without colliding with other obstacles. The test oracle's passing condition (i.e., not resulting in a violation) for collision detection is defined as follows:

$$\underset{1\leq j\leq t}{\forall}(D_c(p_j^E, p_j^{O_k}) > 0 \vee OnBoundary(p_j^{O_k}) \vee collision^{type} = \text{“rear-end”}) \tag{7}$$

where t is the total duration of the scenario, and $D_c(p_j^E, p_j^{O_k})$ is a function that calculates the *shortest distance* between the position of the ego car p_j^E and the position of the k^{th} obstacle $p_j^{O_k}$ at timestamp j. The distance is measured, in meters, between the closest two points between the ego car's polygon and an obstacle polygon. If function D_c returns a non-zero distance in meters between the ego car and any other obstacle during the entire scenario, this indicates a passing condition. If a collision does occur (i.e., D_c returns a distance equal to or less than zero), we further check (1) if the obstacle is in the middle of a lane change maneuver by checking if the obstacle is on a lane boundary ($OnBoundary(p_j^{O_k})$) and (2) if the ego car is being rear-ended by another obstacle. This design effectively eliminates cases where the obstacle ignores the safe distance needed during a lane change and lane follow maneuvers and cause collisions which the ego car cannot be reasonably responsible for [78], [79], [80].

The **Speeding Detection** oracle checks if the ego car reaches its final destination without exceeding the speed limit. The test oracle's passing condition (i.e., not resulting in a violation) for speeding detection is defined as follows:

$$\underset{1\leq j\leq t}{\forall} D_s(s_j^l, s_j^E) \leq \sigma_{safe} \tag{8}$$

where t is the total duration of the scenario, and $D_s(s_j^l, s_j^E)$ is a function that calculates the difference between the ego car's speed s_j^E and the speed limit of the current lane s_j^l at timestamp j. σ_{safe} represents the allowed threshold for an ego car to drive *above* the current speed limit. We allow the ego car to exceed the current speed limit by a maximum of **8** km/h, anything above that is considered a speed violation. We allow some degree of driving above the speed limit since it can be unsafe for the ego car to drive below or at the speed limit in certain conditions [81] (e.g., driving at the speed limit when other cars are going much faster can be dangerous).

The **Unsafe Lane-Change** oracle checks if the ego car reaches its final destination without exceeding a time limit δ_{safe} when changing lanes. Recall from Section IV.B.2, that δ_{safe} represents a safe lane-change duration, which averages at 5 seconds [46]. The test oracle's passing condition (i.e., not resulting in a violation) for unsafe lane change is defined as follows:

$$\underset{1\leq j\leq t}{\forall} c_j^E \leq \delta_{safe} \tag{9}$$

where t is the total duration of the scenario, and c_j^E represents the duration an ego car spends driving between two lanes at timestamp j. If c_j^E at a given time exceeds δ_{safe}, this indicates the occurrence of an unsafe lane change.

The **Fast-Acceleration** oracle checks if the ego car reaches its final destination without causing a rider's discomfort by accelerating too fast. The test oracle's passing condition for fast acceleration is defined as follows:

$$\underset{1\leq j\leq t}{\forall} a_j^E \leq \alpha_{fast} \tag{10}$$

where t is the total duration of the scenario, and a_j^E is the acceleration of the ego car at timestamp j. α_{fast} represents the maximum acceleration allowed for an ego car before it violates a rider's comfort. We allow the ego car to accelerate to a maximum of **4** m/s^2, a threshold utilized in prior work [82] and set by Apollo developers [35].

The **Hard-Braking** oracle checks if the ego car reaches its final destination without causing a rider's discomfort by braking suddenly and excessively. The test oracle's passing condition for hard braking is defined as follows:

$$\underset{1\leq j\leq t}{\forall} a_j^E \geq \alpha_{hard} \tag{11}$$

where t is the total duration of the scenario, and a_j^E is the acceleration of the ego car at timestamp j. α_{hard} represents the minimum acceleration allowed for an ego car before it violates a rider's comfort. We allow the ego car to decelerate to a minimum of $-$**4** m/s^2, a threshold used in prior work [82] and set by Apollo's developers [35].

We differentiate between fast acceleration and hard braking in our fitness functions and oracles for the following reasons: (i) Both violations have conflicting fitness functions. As described in Section IV.B.2, the fitness function for fast acceleration aims to maximize the ego car's acceleration, while the hard-braking fitness function aims to minimize its acceleration, making it critical to highlight the differences between both violations. (ii) Differentiating between hard braking and aggressive acceleration enables us to draw correlations between other safety violations and comfort violations. For example, understanding the correlation between collision and hard braking as opposed to collision and aggressive acceleration. (iii) Having two separate oracles allows customizing the specified thresholds. For

TABLE III
THE SET OF FEATURES, SELECTED FOR EACH VIOLATION TYPE, AND USED BY THE *DUPLICATE VIOLATION DETECTOR* TO CLUSTER SIMILAR VIOLATIONS TOGETHER

Grading Metric	Extracted Features
Collision Detection	$\{p^E_{t_c}, s^E_{t_c}, collision^{type}, T_{O_k}, Z_{O_k}, s^{O_k}_{t_c}\}$
Speeding Detection	$\{p^E_{t_s}, s^E_{t_s}, duration, value_{t_s}\}$
Unsafe LaneChange	$\{p^E_{t_u}, s^E_{t_u}, duration_{t_u}\}$
Fast Acceleration	$\{p^E_{t_f}, s^E_{t_f}, duration, value_{t_f}\}$
Hard Braking	$\{p^E_{t_h}, s^E_{t_h}, duration, value_{t_h}\}$

example, one might want to set a hard braking threshold to be $-$**5** m/s^2 while keeping fast acceleration at **4** m/s^2.

F. Duplicate Violations Detector

One of the challenges of scenario-based testing is the possibility of producing driving scenarios with *similar* violations. To improve the effectiveness of test generation, the *Duplicate Violations Detector* automates the process of identifying and eliminating duplicate violations; it achieves this by using an unsupervised *clustering* technique [83] to group driving scenarios, with similar violations, according to specific *features*.

The set of features used by the clustering algorithm is extracted from the recorded files (Section IV-D). For a **collision** violation, the *Duplicate Violations Detector* extracts 6 features at time t_c, where t_c indicates the first timestamp at which a collision occurs. These features include the location of the ego car $p^E_{t_c}$; ego car's speed $s^E_{t_c}$; $collision^{type}$, which indicates where a collision occurs in respect to the ego car (e.g., "rear-end", "front", "left", etc.); the type of the obstacle (O_k) that collided with the ego car T_{O_k}; the obstacle's size Z_{O_k}; obstacle's speed at collision time $s^{O_k}_{t_c}$.

For the remaining violations, we extract their respective features at times t_s, t_u, t_f, and t_h, which correspond to the first timestamp a **speeding**, **unsafe lane change**, **fast acceleration**, and **hard braking** occurs, respectively. These features include the ego car E's location at a violation time p^E, the speed s^E of E, the length of time for which a violation lasts ($duration$), and the violation $value$.

Given the extracted representation of driving violations in Table III, *Duplicate Violations Detector* clusters driving scenarios with similar violations into groups. For the clustering itself, we chose DBSCAN (i.e., density-based spatial clustering of applications with noise) [83], since it is more suited for spatial data. We also experimented with k-means, which resulted in clusters of undesired structure and quality. We avoided the use of hierarchical clustering [84] due to its computationally expensive nature.

Existing work has suggested using clustering techniques to automatically categorize traffic scenarios or driving behaviours [85], [86], [87], [88], [89]. These approaches are geared towards clustering real-time, multi-trajectory, and multivariate time series data into similar driving encounters or scenario types. Unlike these techniques, SCENORITA aims to eliminate duplicate violations by clustering scenarios with violations. Hence, *Duplicate Violations Detector* only requires a carefully-selected, smaller number of features involving just a few time frames in a scenario.

We did not compare the duplicate elimination component in SCENORITA against DeepHyperion [90] since the latter approach only computes the similarity of road segments based on their smoothness (i.e., smoothness of turns), complexity (i.e., number of turns), and orientation (i.e., number of directions). SCENORITA computes the similarity of reported violations (collision, speed, hard braking, etc.), which is very different from computing the similarity of road segments like in DeepHyperion.

Duplicate elimination in AsFault [25] is based on similar road segments, where the similarity between roads is calculated using the Jaccard index. In AV-FUZZER [24], duplicate elimination is based only on obstacles' trajectories, where the similarity between trajectories is calculated using the Euclidean distance. AutoFuzz [30] detects duplicate traffic violations using a learning-based seed selection and mutation strategy. The paper claims to detect many unique violations. However, after rerunning the approach and inspecting the scenarios generated, it is evident that the evaluation of AutoFuzz has been significantly impacted by simulator-induced problems. We will elaborate on this further in Section V-B.

V. EVALUATION

In order to empirically evaluate SCENORITA, and to understand how its configuration affects the quality of generated tests, we investigate the following research questions:

RQ1: How effective are SCENORITA's generated driving scenarios at exposing AV software to safety and comfort violations?
RQ2: What is the runtime efficiency of SCENORITA's generated tests and oracles?
RQ3: How diverse are scenarios generated by SCENORITA?
RQ4: To what extent does SCENORITA eliminate duplicate violations?

A. Experiment Settings

Our extensive evaluation consists of executing 79,051 virtual tests on Baidu Apollo. For this reason, we conducted our experiments on 4 machines: 2 machines each with 2 AMD EPYC 7551 32-Core Processor and 512GB of RAM, and another 2 machines each with 1 Core i9 16-Core Processor and 128GB of RAM. All 4 machines are running Ubuntu 20.04. In the current implementation of SCENORITA, we focus our efforts on testing Baidu Apollo 7.0 [91], an open-source and production-grade AV software system that supports a wide variety of driving scenarios and explicitly aims for both safety and rider's comfort.

We use Apollo's simulation feature, `Sim-Control`, to simulate driving scenarios. `Sim-Control` [92] does not simulate the *control* of the ego car; instead, the ego car acts on the

planning results. `Sim-Control` takes Apollo's planning decisions $\mathcal{PD}_E$ for ego car E (Definition 2) as input and publishes localization messages corresponding to the planned positions $\mathbb{P}_{\mathcal{P}}$. The planning module then makes the next planning decision based on the updated location of the ego car, and the cycle continues to simulate a driving scenario. `Sim-Control` is also recommended by one of Apollo's maintainers to use when focusing testing on the planning module [93]. By using this built-in simulation capability, SCENORITA does not rely on an external simulator and, thus, does not have to conform to how the simulator specifies a scenario, allowing SCENORITA to take full advantage of our novel, fully-mutable gene representation.

We configured SCENORITA to generate driving scenarios for 4 high-definition maps of cities/street blocks located in California: Sunnyvale is a large map consisting of 3,061 lanes, with a total length of 107 km; San Francisco is another large map with 1,524 lanes and a total length of 109 km; San Mateo is a medium map with 1,305 lanes and a total length of 24 km; and Borregas Ave, which is a small map of a city block in Sunnyvale with 60 lanes and a total length of 3 km. The 4 maps consist of various types of road curvature (e.g., straight, curved, intersections) and different types of lanes (e.g., highways, city roads, bike lanes, etc.). California is at the center of AV research, and one of the main deployment grounds for AV; Waymo, GM Cruise, and 60 other companies have obtained commercial licenses for testing AVs in CA [94]. As a result, these maps are highly representative of real-world AV driving scenarios with a wide variety of diverse environmental elements. Note that any high-autonomy (L4) ADS requires HD maps.

We believe that bugs in AV software can be exposed not only by placing certain obstacles with certain attributes near the ego car, but also by generating tests that cover as many different parts of the map, and subsequently different road setups (intersection, u-turn, multi-lane roads, etc.), as possible. For example, A large-scale study was conducted on 60,000 collisions in Orange County, California between 2010 and 2018, to determine which intersections pose the most risk for drivers in Orange County [95]. This study ranked certain intersections of cities in Orange County based on a Crash Risk Index (CRI) score, i.e., a composite score that weighs the volume of collisions and severity of injuries. The study found that certain intersections, such as Alicia Parkway and Jeronimo Road—the only crossing in Mission Viejo (a city in Orange County) to make the list—saw the most injuries and the third-highest number of crashes during the study period.

Similarly, in October 2021, a Pony.ai vehicle operating in autonomous mode hit a street sign on a *median*, i.e., the strip of land between the lanes of opposing traffic on a divided highway, in Fremont, California, prompting California to suspend the company's driverless testing permit; Pony.ai said that the crash occurred less than 2.5 seconds after the automated driving system shut down. It said "in very rare circumstances, a planning system diagnostic check could generate a false positive indication of a geolocation mismatch" [96].

These studies and incidents further emphasize the need to consider as many different parts of the map as possible, as different road setups might expose more bugs or result in more violations. We further discussed in Section V-B the impact of each map on the found violations, and how SCENORITA with its fully mutable obstacle representation was more effective in finding violations, especially when running tests on larger and more complex maps.

We do not compare SCENORITA against certain approaches due to the fact that they address a different research problem. For example, [25] and [61], [62], [63], are merely concerned with generating road networks, [26] is concerned with finding tests for which the ego car can avoid if traveling on an alternative path, etc. The most closely related approaches [24], [30] depend on a simulator (i.e., SVL [97]) external to the ADS and has been sunsetted [98], making it no longer available from its official provider. Due to the aforementioned problem, we spent extra effort to reverse engineer a closed-source key component of SVL and have since maintained the availability of SVL for the benefit of the research community [99]. In addition, the original implementation of AV-FUZZER maintained by the authors [100] depends on a specific version of SVL that is outdated and even has missing components, preventing it from being reused. As a result, we rely on a forked version of AV-FUZZER [101], which can operate with the most recent version of SVL and Apollo 7.0 [91], and includes reimplementations of the missing components. With these challenging reusability issues addressed, we are now able to facilitate direct comparison between SCENORITA, AV-FUZZER [24], [101], and AutoFuzz [30], [102].

For each generation, there are 30 scenarios (i.e., demes), each with a minimum of 10 obstacles (i.e., individuals) and a maximum of 30 obstacles. Note that the higher number of obstacles there are, the more overhead there is to generate the real-time trajectories for all of the obstacles, and the scenario may become overly crowded on a small map. We configured the maximum scenario duration to be 30 seconds and stopped scenario generation after 12 hours. We used the crossover operator with a probability of 0.8 and mutated single individuals with a probability of 0.2. Mutating a scenario by either adding a new obstacle from another scenario or removing an obstacle was performed with a probability of 0.1 each. we followed the guidelines in [103], [104]—which suggests that standard parameter settings are usually recommended—leading us to use default settings in DEAP-1.3 [105], the framework used in our search-based implementation. To compare AV-FUZZER with SCENORITA, we rerun AV-FUZZER [101] for 12 hours and analyzed scenarios generated by AV-FUZZER [101] using metrics defined in Section IV-E. We follow a similar setup to compare AutoFuzz [102] with SCENORITA; however, using the configuration suggested by the authors of AutoFuzz [106], [107], we are not able to run experiments for longer than 1 hour [108], [109], and we are not able to use AutoFuzz to generate scenarios on San Francisco because the simulator kept exiting after the first scenario. Additionally, we also evaluate SCENORITA's performance on maps that none of the other state-of-the-art approaches support (i.e., San Mateo and Sunnyvale). To account for the non-determinism of approaches, every experiment (i.e., one approach spending a maximum of 12 hours generating scenarios on one map) is repeated 5 times, resulting

TABLE IV
A COMPARISON OF BUG-REVEALING VIOLATIONS PRODUCED BY SCENORITA, AV-FUZZER, AND AUTOFUZZ

Map	San Francisco											Borregas Ave											Sunnyvale Loop					San Mateo				
Approach	SCENORITA					AV-FUZZER					p	$\hat{A_{12}}$	SCENORITA					AutoFuzz					p	$\hat{A_{12}}$	SCENORITA					SCENORITA				
Experiment No.	1	2	3	4	5	1	2	3	4	5			1	2	3	4	5	1	2	3	4	5			1	2	3	4	5	1	2	3	4	5
Collision (duplicates)	15	2	2	1	2	0	0	0	0	0	0.01	1.0	7	8	7	11	7	1	0	2	0	0	0.01	1.0	4	1	3	1	0	15	1	2	14	19
Collision (unique)	10	1	1	1	1	0	0	0	0	0	0.01	1.0	5	6	3	7	6	1	0	1	0	0	0.01	1.0	3	1	1	1	0	3	1	1	5	5
Speeding (duplicates)	5	19	13	17	8	0	0	0	0	0	0.01	1.0	37	0	1	0	4	0	0	0	0	0	0.07	0.8	34	132	101	92	31	148	171	108	110	57
Speeding (unique)	4	6	6	2	6	0	0	0	0	0	0.01	1.0	6	0	1	0	3	0	0	0	0	0	0.07	0.8	8	25	22	16	9	23	20	17	25	9
FastAccel (duplicates)	19	62	43	41	35	0	0	0	0	0	0.01	1.0	1	0	0	0	1	0	0	0	0	0	0.17	0.7	1	3	0	1	3	0	5	2	3	1
FastAccel (unique)	8	14	16	8	12	0	0	0	0	0	0.01	1.0	1	0	0	0	1	0	0	0	0	0	0.17	0.7	1	2	0	1	1	0	3	1	1	1
HardBrake (duplicates)	165	311	275	187	219	0	4	0	0	0	0.01	1.0	109	8	55	23	34	0	0	0	0	0	0.01	1.0	1	23	12	5	0	10	31	7	31	8
HardBrake (unique)	29	35	9	23	24	0	4	0	0	0	0.01	1.0	21	6	18	9	8	0	0	0	0	0	0.01	1.0	1	12	10	3	0	3	10	3	13	3
USLC (duplicates)	240	190	193	114	115	0	0	0	3	1	0.01	1.0	411	137	217	171	209	22	20	19	15	11	0.01	1.0	27	31	32	12	33	92	91	89	124	144
USLC (unique)	54	62	56	16	27	0	0	0	2	1	0.01	1.0	52	37	50	30	35	11	5	10	5	4	0.01	1.0	2	10	10	7	9	13	30	14	16	35
Total (duplicates)	444	584	526	360	379	0	4	0	3	1	0.01	1.0	565	153	280	205	255	23	20	21	15	11	0.01	1.0	67	190	148	111	67	265	299	208	282	229
Total (unique)	105	118	88	50	70	0	4	0	2	1	0.01	1.0	85	49	72	46	53	12	5	11	5	4	0.01	1.0	15	50	43	28	19	42	64	36	60	53

in a total of 30 experiments and 305 hours of test executions. Given the same amount of time, SCENORITA, AV-FUZZER, and AutoFuzz generated a total of 75,900, 2,651, and 500 scenarios, respectively.

B. RQ1: SCENORITA's Effectiveness at Generating Violation-Inducing Scenarios

RQ1 investigates whether SCENORITA leads to more reported violations, compared to other state-of-the-art approaches. In conducting our evaluation of this research question, we followed the guidelines in [104] for comparing SCENORITA against both AV-FUZZER [24] and AutoFuzz [30]. Hence, we performed two statistical tests: 1) *Mann-Whitney U-test p-values* to determine statistical differences, and 2) *Vargha–Delaney's* $\hat{A_{12}}$ *index* [110] to determine the effect size. The results of these tests, in our experiments, are interpreted as follows: If Mann-Whitney U-test produces $p \leq 0.05$, this indicates that there is a significant difference between the quality of solutions provided by SCENORITA and AV-FUZZER or AutoFuzz. The $\hat{A_{12}}$ statistical test measures how often, on average, one approach outperforms another; if $\hat{A_{12}} = 0.5$, then the two approaches achieve equal performance; if $\hat{A_{12}} > 0.5$, then the first approach is better; otherwise, the first approach is worse. The closer $\hat{A_{12}}$ is to 0.5, the smaller the difference between the techniques; the further $\hat{A_{12}}$ is from 0.5, the larger the difference.

Table IV summarizes the number of violations and the number of *unique* violations found by each approach. From Table IV, we observe that SCENORITA found a total of 1,146 unique violations in all maps over the course of our experiments. Furthermore, we find that SCENORITA discovered 60.57 times more *unique* violations compared to AV-FUZZER ($\hat{A_{12}} = 1.0, p < 0.05$), and 7.24 times more *unique* violations compared to AutoFuzz ($\hat{A_{12}} = 1.0, p < 0.05$).

SCENORITA consistently detects more collision violations on Borregas Avenue (avg. $= 5.4, \sigma = 1.36$) compared to that on other maps. We believe the size of the map correlates to the number of collision violations detected, as it increases the chance of the obstacle being closer to the ego car during a scenario. On larger maps (e.g., Sunnyvale Loop, San Francisco), obstacles may be traveling on different parts of the map that are far away from the ego car, and evolve for many generations before they come near the ego car.

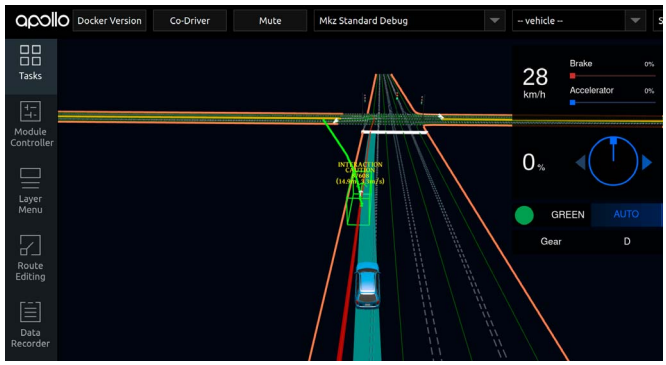


Fig. 7. A scenario generated by SCENORITA that results in **Apollo rear-ending a dynamic obstacle**. The planning module does not plan to slow down for the obstacle because of the incorrect prediction trajectory.

Example Scenario. Fig. 7 shows a snapshot of a scenario before Apollo and an obstacle collided on the map of San Francisco. In this scenario, Apollo ends up colliding with the obstacle from behind. Since the obstacle is traveling in its own lane and has the right-of-way, Apollo is fully responsible for this collision violation. After inspecting the violation from Dreamview, we identified its root cause being *incorrect prediction trajectory*. The obstacle is traveling at 3.3 m/s (11.88 km/h) and is continuing to travel in a straight line, but Apollo's prediction module predicts the obstacle will drive off-road toward the left. Based on the incorrect prediction trajectory, Apollo continues to plan to drive at 8.33 m/s (30 km/h) despite being only 7.07 m away from the obstacle. Note that the distance displayed by Dreamview (14.9 m) refers to the distance between the center of the obstacle and the center of the ego car, and 7.07 m is the distance between the rear of the obstacle and the front of the ego car. To avoid this collision, Apollo needs to decelerate at 4.92 m/s^2. Unfortunately, by the time the planning module realizes there will be a collision and starts making stop decisions, it is already too late and Apollo collides into the obstacle from behind. The video recording and its corresponding Apollo record file are available at [111].

AV-FUZZER generates scenarios that focus on the ego car traveling in a straight line while the obstacle frequently changes lanes around the ego car. In a 12-hour experiment, AV-FUZZER generated 570 scenarios and 190 of those scenarios involve the

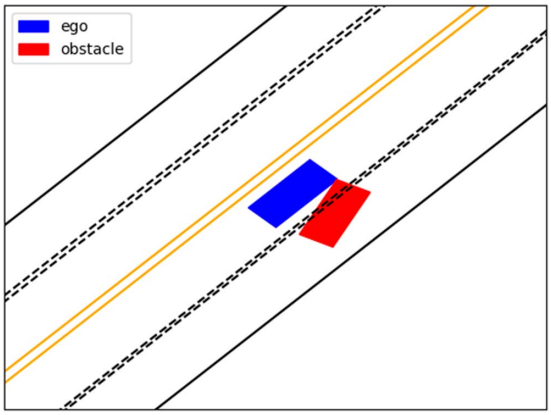


Fig. 8. A scenario generated by AV-FUZZER causing the ADS to be involved in a side-impact collision. The obstacle changes lanes without considering the distance needed to complete such a maneuver.

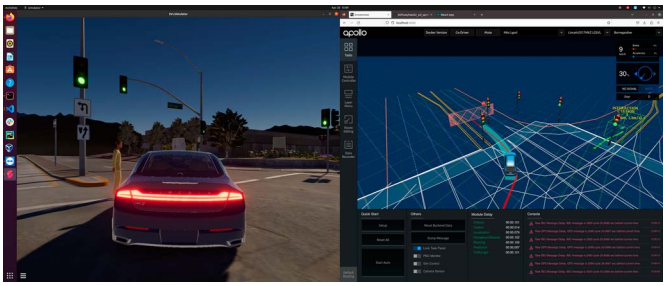


Fig. 9. Screenshot of a scenario generated by AutoFuzz using the version of the simulator (left) specified in the paper with perception ground truth; the right side shows Apollo's Dreamview visualizing ADS' execution state, which is missing a yellow polygon that represents a pedestrian.

collision between the ego car and an obstacle. However, 69 of the collisions detected are obstacles rear-ending the ego car, and the remaining 121 collisions occurred because the obstacle recklessly changes lanes. As shown in Fig. 8, a collision between the ego car and the obstacle occurs during the obstacle's lane change maneuver; however, the ADS is not responsible for such collision as the obstacle does not consider the distance needed to complete such a maneuver. SCENORITA's design of its collision oracle effectively eliminates these types of collisions as the obstacle is always fully responsible, i.e., there is no bug or misbehavior causing such collisions in the ADS.

For AutoFuzz, we were not able to reproduce similar results described in the paper. A careful inspection and analysis reveal that the simulator is responsible for causing the ADS to collide with obstacles. First, the simulator has known issues when working with Apollo's perception module, causing flickering obstacle detection results [112], [113]; second, when solving the previous problem by using the ground truth perception provided by the simulator, pedestrians cannot be detected [114], [115], [116]. Collisions detected under an incorrect SVL configuration cannot effectively reveal ADS bugs, and when rerunning AutoFuzz with the simulator's ground truth perception, we only observe one type of valid collision, where the ego car stopped in a junction to yield right-of-the-way to the obstacle. The obstacle then crashes into the ego car because the obstacle only travels along a straight line at a constant speed and does not react

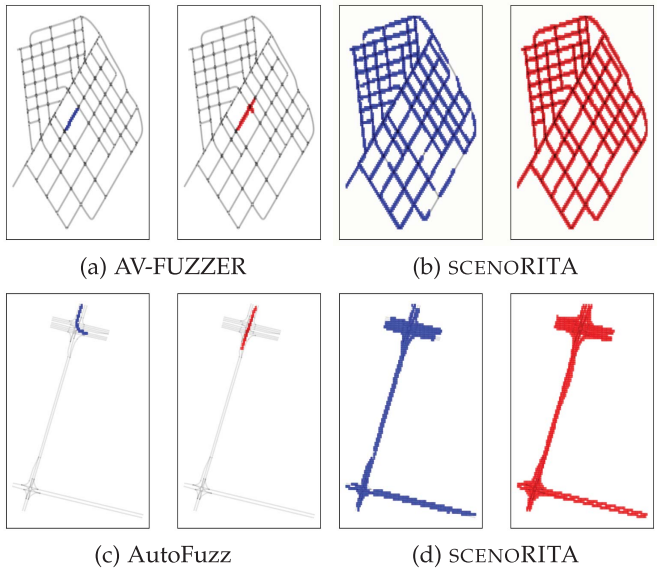


Fig. 10. Parts of the map covered by each approach: blue indicates coverage by the ego car, red indicates coverage by obstacles, and black lines are lanes on the map.

to the ego car, which has already stopped. Fig. 9 illustrates the problem with the simulator's ground truth perception, as the pedestrian's position is not provided to Apollo, causing a collision in the simulator.

Finding 1: In our experiments, SCENORITA found a total of **1,146** *unique* safety and comfort violations including: **62** collisions, **208** speed violations, **565** unsafe lane changes, **71** fast acceleration violations, and **240** hard-braking violations. Overall, SCENORITA significantly and always ($\hat{A}_{12} = 1.0$, $p < 0.05$) outperforms other state-of-the-art approaches.

C. RQ2: Diversity of SCENORITA Scenarios

RQ2 investigates whether scenarios generated by SCENORITA are more diverse, compared to other state-of-the-art approaches. Fig. 10(a) depicts positions in which either the ego car or obstacles have been on the map in scenarios generated by AV-FUZZER, and Fig. 10(b) depicts scenarios generated by SCENORITA. Scenarios generated by AV-FUZZER require manual specification, thus generating scenarios that are limited to only the manually-specified parts of the map. When using a map with 1,524 lanes, AV-FUZZER only generates scenarios with the ego car and obstacles traveling on a single lane, limiting its capability to test other types of ADS maneuvers in more complex road structures and junctions; SCENORITA, with its fully mutable and more fine-grained representation of obstacles, is capable of generating diverse scenarios in which the ego car and obstacles can be at any point on the map. Similarly, Fig. 10(c) and Fig. 10(d) compares the positions the ego car and obstacles cover on SCENORITA and AutoFuzz.

Covering more parts of the map allows the ADS to be tested in more complex scenarios that involve junctions and traffic control devices. While San Francisco is a large map with 91

TABLE V
TIME EFFICIENCY (IN SECONDS) OF SCENORITA PER SCENARIO/GENERATION

HD Maps	Stats Per Scenario				Stats Per Generation			
	Generate	Play	Analyze	E2E	Mut/Cx	Evaluate	Select	E2E
Borregas Ave	2.53	42.29	11.75	56.57	0.31	236.13	0.01	236.45
San Francisco	5.30	42.68	23.55	71.53	3.57	262.10	0.04	265.72
San Mateo	14.18	42.55	16.34	73.08	6.16	484.20	0.00	490.36
Sunnyvale Loop	25.06	42.34	17.57	84.96	14.73	807.35	0.06	822.13

junctions, 201 traffic signals, and 9 stop signs, AV-FUZZER covers none of these map elements (0%) because it is specified to cover only a single straight lane without passing through any traffic control devices; at the same time, SCENORITA covers 91 (100%) junctions, 177 (88.06%) traffic signals, and 4 (44.44%) stop signs. While Borregas Avenue is a small map with only 2 junctions, 4 traffic signals, and 2 stop signs, scenarios generated by AutoFuzz can only cover 1 junction and 1 signal at that junction; when using the same map, SCENORITA covers all junctions, signals, and stop signs. Manual specification of a different scenario type for AV-FUZZER and AutoFuzz to increase their map coverage would require an extensive amount of error-prone, time-consuming human effort to (1) specify the same number of obstacles, (2) cover similar parts of the map, and (3) cover a similar number of junctions and traffic control devices compared to SCENORITA.

SCENORITA's ability to automatically identify routing requests and cover significantly more parts of the map also allowed us to identify rare cases where the ego car stops indefinitely at certain parts of the map. We manually analyzed the scenarios and were able to localize the root cause in the HD map. We submitted a fix for this bug and it has been accepted by Apollo's developers [117].

Finding 2: SCENORITA is able to cover substantially more parts of the map through (1) its more general design of analyzing the map and automatically specifying paths of obstacles, (2) a fully mutable genetic representation that allows obstacles' paths to be different across scenarios, and (3) using demes to construct scenarios in which the ego car starts and ends at different locations. On a large map with 91 junctions, 201 traffic signals, and 9 stop signs, the state-of-the-art approaches cover none of these, while SCENORITA covers 91, 177, and 4 of them, respectively.

D. RQ3: Efficiency of SCENORITA

In RQ3, we study the efficiency of SCENORITA by measuring its execution time per scenario, per generation, and the number of scenarios generated per 12-hour period.

Table V shows that the efficiency of SCENORITA correlates to the size of the HD Map used. On the smallest map, SCENORITA takes **56.57** seconds, on average, to execute a scenario from end-to-end (*E2E*); it takes **2.53** seconds, on average, to generate the scenario representation, confirm its validity according to the *domain-specific constraints*, and generate the obstacles' trajectories (*Generate*); **42.29** seconds to initialize the ADS and play the corresponding driving simulation (*Play*); and **11.75** seconds for checking the grading metrics (*Analyze*). Simulating driving scenarios is time-consuming (e.g., properly resetting the states of the ADS, playing the simulation for 30 seconds, and recording the ego car's behavior). As a result, the scenario simulation stage strongly affects the efficiency of the overall test generation process in SCENORITA. To leverage the insight that multiple scenarios can be parallelized using the lightweight simulation (i.e., `Sim-Control`), each generation of the genetic algorithm takes less time compared to sequentially executing the scenarios. We find that the time consumed by producing offspring (*Mut/Cx*) and performing selection (*Select*) is negligible, and it takes, on average, 236.45 seconds to evaluate 30 scenarios on the smallest map (an 86.07% reduction compared to sequential execution time), and 822.13 seconds on the largest map (a 67.74% reduction compared to sequential execution time).

TABLE VI
AVERAGE NUMBER OF SCENARIOS GENERATED PER HOUR

Map \ Approach	AV-FUZZER	AutoFuzz	SCENORITA
Borregas Ave	-	76.9	457.5
San Francisco	44.2	-	407.5
San Mateo	-	-	222.5
Sunnyvale Loop	-	-	132.5

Table VI shows a comparison between the number of scenarios generated per hour for SCENORITA, AV-FUZZER, and AutoFuzz. On San Francisco, a map that both SCENORITA and AV-FUZZER support, SCENORITA generated 407.5 scenarios per hour, outperforming AV-FUZZER by 821.95%. Similarly, on Borregas Avenue, SCENORITA outperforms AutoFuzz by 494.93%. On the maps that other state-of-the-art approaches do not support, SCENORITA generated scenarios at 222.5 per hour and 132.5 per hour, respectively, on San Mateo and Sunnyvale Loop.

Finding 3: SCENORITA efficiently mitigates the overhead of initializing the ADS and analyzing scenarios, and generates 30-second driving scenarios that expose AV software to safety and comfort violations at a rate of 2.19 to 7.61 scenarios per minute. Overall, SCENORITA is between 4.95 and 8.22 times more efficient compared to other state-of-the-art approaches.

E. RQ4: Duplicate Violation Detection

This RQ investigates the extent to which SCENORITA eliminates similar violations. To answer this RQ, we configure DBSCAN [83] to cluster the scenarios with similar violations into the same group, based on a set of features as described in Section IV-F. We adopted the approach in [118] to automatically determine the optimal value for *epsilon*; *epsilon* defines the

maximum distance allowed between two points within the same cluster. Eliminating duplicate violations is quick and takes, on average, 0.1 seconds per experiment.

To confirm the correctness of generated clusters, three of the authors manually and independently evaluated the accuracy of generated clusters in 10 randomly-selected experiments (i.e., one approach spending a maximum of 12 hours generating scenarios on one map) out of a total of 30 (33.33%). The authors examined violations in the same clusters to confirm whether they are similar by comparing a set of features associated with each scenario in the cluster. For example, consider two scenarios, `Scenario_1` and `Scenario_17`, both of which are in the same cluster and have a collision violation: In such a case, we compare the set of features (from Section IV-F) of the two scenarios to confirm that the collision occurred in the same position in `Scenario_1` as it did in `Scenario_17` ($p^E_{t_c}$), the ego car in both scenarios collided with an obstacle with the same type (T_{O_k}) and size (Z_{O_k}), the crash in both scenarios is the same ($collision^{type}$), etc. The authors also replayed these scenarios on Dreamview to observe if the scenarios in one cluster have similar violations.

Table IV shows all violations (including duplicates) generated by SCENORITA, AV-FUZZER, and AutoFuzz along with the unique number of violations (generated by the *Duplicate Violations Detector*). From the results in Table IV, we observe that SCENORITA generated a total of 431 *unique* violations on San Francisco compared to AV-FUZZER which generated 7 *unique* violations; SCENORITA also generated a total of 305 *unique* violations on Borregas Avenue compared to AutoFuzz which generated 37. On two other maps that none of the state-of-the-art approaches support, SCENORITA generated a total of 155 and 255 *unique* violations, respectively.

Finding 4: Our manual verification of 33.33% of our experiments shows that *Duplicate Violation Detector* is able to identify and eliminate duplicate violations. From a total of 5,617 violations detected by SCENORITA, it effectively and correctly clustered all violations generated into 1,146 *unique* violations. When compared against state-of-the-art approaches, SCENORITA finds at least 7 times more *unique* safety and comfort violations on maps they support.

VI. Threats to Validity

Internal threats. One potential threat to internal validity is the selection of scenario durations: Simulation-based tests require the execution of time-consuming computer simulations to produce violations. We determined from our experimentation that our selected scenario duration of 30 seconds finds a significant number and variety of violations without incurring drastically long test execution times. Furthermore, there is no agreed-upon threshold in related work that dictates the correct scenario duration. For example, [69] opted for a duration time of 10 seconds per scenario since it serves the goal of their test generation framework.

Another threat to validity is related to choosing DBSCAN (i.e., density-based spatial clustering of applications with noise) in eliminating duplicate violations. To mitigate this threat to validity, we ran both k-means and DBSCAN on a random set of scenarios with duplicate violations, then manually inspected the clusters of scenarios with similar violations generated by both techniques. We found that (i) k-means resulted in clusters of undesired structure and quality; (ii) unlike DBSCAN, k-means is not an ideal algorithm for latitude-longitude spatial data because it minimizes variance, not geodetic distances; (iii) DBSCAN is deterministic while k-means is not; and (iv) DBSCAN does not require specifying the number of clusters in advance—it determines them automatically based on the maximum distance allowed between two points within the same cluster. We avoided using hierarchical clustering due to its computationally expensive nature.

SCENORITA's use of the ADS's lightweight built-in simulation capability also poses an internal threat. `Sim-Control` does not model the physics of traffic participant collision. As a result, the ego vehicle may continue to move forward after a collision since there is no physics engine preventing 3D objects from overlapping with each other. This characteristic of `Sim-Control` does not invalidate any of the violations but instead reveals the ADS' planning module continues to plan on moving forward, even after being involved in a collision. SCENORITA mitigates the threat by discarding events that occurred after any type of collision occurring between the ego car and the obstacles since their trajectories will be different in the physical world.

Another threat to validity arises from verifying the correctness of the oracles implemented and scenarios generated by SCENORITA. There are, unfortunately, neither automated strategies nor ground truth that can be used to assess the accuracy. For that reason, we manually verified all of the scenarios that involve collision violations and a randomly sampled set of scenarios for other types of violations. To further mitigate this threat, we make our testing framework available online [47] for others to verify and build upon our work, thus supporting reusability and reproducibility as well.

To account for validity threats arising from randomness in search algorithms, we follow the guidelines in [104]: (i) we repeated the experiments for each approach (SCENORITA, AV-FUZZER, AutoFuzz) 5 times, (ii) we used the non-parametric Mann-Whitney U-test to detect statistical differences and reported the obtained p-value, and (iii) we reported $\hat{A_{12}}$ index (a standardized effect size measure).

To mitigate threats arising from our selection of search operators, we selected (i) a widely-used algorithm [119] in the search-based software engineering (SBSE) community, i.e., NSGA-II and (ii) crossover and mutation algorithms that best fits our gene representation. We opted for NSGA-II as a selection operator since it's an effective algorithm for solving multi-objective optimization problems (i.e., problems with conflicting fitness functions) and further aims to maintain the diversity of individuals. In other words, we specifically intended to generate scenarios with multiple violations coexisting together. We further find that this design does not result in test smells causing much of an issue. For example, collision does not always trigger hard braking; as a matter of fact, from a

total of 1,514 scenarios with a hard-braking violation, 1,504 (99.34%) of them do not contain a bug-revealing collision violation. For parameter tuning, we followed the guidelines in [103], [104]—which suggests that standard parameter settings are usually recommended—leading us to use default settings in DEAP-1.3 [105], the framework used in our search-based implementation.

External threats. One external threat is that we applied SCENORITA to a single AV software system, Apollo. To mitigate the threat, we selected the only high autonomy (i.e., Level 4), open-source, production-grade AV software system that supports a wide variety of driving scenarios and explicitly aims for both safety and driver comfort. To mitigate threats related to the generalizability of our results to other maps, we applied SCENORITA to three high-definition maps of cities in California: Borregas (60 lanes), San Mateo (1,305 lanes), San Francisco (1,524 lanes) and Sunnyvale (3,061 lanes). Note that Autoware [120], despite being open-source and widely-used [121], is considered a research-grade and not a production-grade AV software system [122], [123], which we further verified through speaking with Christian John, the Vice Chair and Chief Software Architect of Autoware.

Construct Validity. The main threat to construct validity is how we measure and calculate safety and comfort violations. To mitigate this threat, we measure these violations using grading metrics defined by Apollo's developers [35]. We utilize thresholds (e.g., speeding or acceleration thresholds) set by Apollo's developers [35]; the U. S. Department of Transportation [46]; or thresholds used by major AV companies (e.g., Alphabet Waymo [81]).

Another threat to construct validity and a potential limitation to our approach is how we define safety violations. We use a single oracle to detect safety violations that the ADS is responsible for, while some other oracles may also be suited for safety, such as checking if the ego car invades opposing lanes and driving off-road. This threat is mitigated as the ego car, in some cases, may be forced to execute maneuvers that pose safety risks to reach its destination, such as borrowing opposing lanes to bypass static obstacles that are blocking a single-lane road. This type of action should still be allowed when the maneuver can be executed safely without disturbing traffic; if the maneuver could not be executed safely, it will still result in a collision violation which our approach will be able to detect.

Another threat to construct validity involves SCENORITA's choice of simulation. The state-of-the-art approaches that we compared against all generate tests using an external simulator, while SCENORITA generates tests using ADS' built-in simulation capability. Running simulations with an external simulation is more hardware-intensive as it requires the machine to run additional software (i.e., the simulator) and requires the ADS to run both the planning module and the control module, as opposed to only running the planning module when using ADS' built-in simulation. To mitigate this threat, we use high-performance machines (e.g., a machine with 2 32-core processors and 512GB of RAM) to conduct our experiments. Moreover, the built-in simulation capability [93] is recommended by one of the ADS' maintainers since it allows tests to focus on finding bugs in the planning module, which is the main focus of this paper.

VII. Conclusion

In this paper, we propose SCENORITA, a novel search-based testing framework, which exposes AV software to 3 types of safety-critical and 2 types of motion sickness-inducing scenarios in a manner that reduces duplicate scenarios, allows fully mutable obstacles with valid and modifiable obstacles trajectories, and follows domain-specific constraints obtained from authoritative sources. We evaluate SCENORITA on Baidu Apollo, a high autonomy (L4), open-source, and production-grade AV software system that supports a wide variety of driving scenarios.

We compare SCENORITA with the state-of-the-art testing approaches that only use a partially mutable representation. SCENORITA found a total of 1,146 *unique* safety and comfort violations including: 62 collisions, 208 speed violations, 565 unsafe lane changes, 71 fast-acceleration violations, and 240 hard-braking violations—all of which are bug-revealing. SCENORITA is able to cover all junctions on both maps and at least 86.19% of all traffic control devices, while state-of-the-art approaches can cover at most 1. Moreover, SCENORITA is more efficient as it generates between 4.95 and 8.22 times more scenarios per hour compared to other state-of-the-art approaches.

For future work, we aim to expand SCENORITA to handle (i) the generation of scenarios and oracles focused on traffic lights and stop signs and (ii) extending the work to other AV software systems (e.g., Autoware).

VIII. Acknowledgment

The authors would like to thank the anonymous reviewers for their constructive feedback. The authors gratefully acknowledge the support of NSF.

References

[1] Waymo. Accessed: May, 2023. [Online]. Available: https://waymo.com/
[2] M. Wayland, "Ford unveils new self-driving test vehicle for 2022 launch," *CNBC*, Aug. 2021. [Online]. Available: https://cnb.cx/3zi2AIQ
[3] "Waymo and Intel collaborate on self-driving car technology." Intell. Accessed: May, 2023. [Online]. Available: https://www.intc.com/news-events/press-releases/detail/205/waymo-and-intel-collaborate-on-self-driving-car-technology
[4] A. J. Hawkins, "Waymo's autonomous cars have driven 8 million miles on public roads," *Verge*, Aug. 2021. [Online]. Available: https://bit.ly/43uSNwb
[5] A. J. Hawkins, "You can take a ride in a self-driving Lyft during CES," *Verge*, Aug. 2021. [Online]. Available: https://bit.ly/3acRidI
[6] "Baidu starts mass production of autonomous buses." Deutsche Welle. Accessed: May, 2023. [Online]. Available: https://bit.ly/3oJXrSy
[7] M. Bertoncello and D. Wee, "Ten ways autonomous driving could redefine the automotive world," *McKinsey*, vol. 6, 2015. [Online]. Available: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/ten-ways-autonomous-driving-could-redefine-the-automotive-world
[8] "Tesla Autopilot." Tesla. Accessed: May, 2023. [Online]. Available: https://www.tesla.com/autopilot
[9] "Uber AV: Autonomous mobility and delivery." Uber. Accessed: May, 2023. [Online]. Available: https://www.uber.com/us/en/autonomous/
[10] "Self-driving Uber car kills pedestrian in Arizona, where robots roam," *New York Times*, Mar. 2018. [Online]. Available: https://www.nytimes.com/2018/03/19/technology/uber-driverless-fatality.html

[11] J. Horwitz and H. Timmons, "There are some scary similarities between Tesla's deadly crashes linked to autopilot," *Quartz*, Sep. 2016. [Online]. Available: https://bit.ly/3Zppqsz

[12] R. Felton, "Two years on, a father is still fighting Tesla over autopilot and his son's fatal crash," *Jalopnik*, Feb. 2018. [Online]. Available: https://bit.ly/42OFMxS

[13] "Tesla driver dies in first fatal crash while using autopilot mode," *Guardian*, Apr. 2016. [Online]. Available: https://bit.ly/3nlNavo

[14] "Self-driving Uber car kills pedestrian in Arizona, where robots roam," *New York Times*, Mar. 2018. [Online]. Available: https://nyti.ms/3abv3Vq

[15] "Tesla: Autopilot was on during deadly mountain view crash," *Mercury News*, Sep. 2023. [Online]. Available: https://bayareane.ws/3U1p4av

[16] A. Tilley, "Google's self-driving car caused its first accident," *Forbes,* Feb. 2016. [Online]. Available: https://bit.ly/3acQwgO

[17] "Tesla autopilot system found probably at fault in 2018 crash," *New York Times*, Feb. 2020. [Online]. Available: https://nyti.ms/3qXuioY

[18] S. Khastgir, S. Birrell, G. Dhadyalla, and P. Jennings, "Identifying a gap in existing validation methodologies for intelligent automotive systems: Introducing the 3XD simulator," in *Proc. IEEE Intell. Veh. Symp. (IV)*. Piscataway, NJ, USA: IEEE, 2015, pp. 648–653.

[19] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE Int. J. Transp. Saf.*, vol. 4, no. 1, pp. 15–24, 2016.

[20] R. Hussain and S. Zeadally, "Autonomous cars: Research results, issues, and future challenges," *IEEE Commun. Surveys. Tuts.*, vol. 21, no. 2, pp. 1275–1313, 2nd Quart. 2019.

[21] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" *Transp. Res. A,* Policy Pract., vol. 94, pp. 182–193, 2016.

[22] P. McMinn, "Search-based software test data generation: A survey," *Softw. Testing, Verification Rel.*, vol. 14, no. 2, pp. 105–156, 2004.

[23] G. Fraser and A. Arcuri, "Evosuite: Automatic test suite generation for object-oriented software," in *Proc. 19th ACM SIGSOFT Symp. 13th Eur. Conf. Found. Softw. Eng.*, 2011, pp. 416–419.

[24] G. Li et al., "AV-FUZZER: Finding safety violations in autonomous driving systems," in *Proc. IEEE 31st Int. Symp. Softw. Rel. Eng. (ISSRE)*. Piscataway, NJ, USA: IEEE, 2020, pp. 25–36.

[25] A. Gambi, M. Mueller, and G. Fraser, "Automatically testing self-driving cars with search-based procedural content generation," in *Proc. 28th ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2019, pp. 318–328.

[26] A. Calò, P. Arcaini, S. Ali, F. Hauer, and F. Ishikawa, "Generating avoidable collision scenarios for testing autonomous driving systems," in *Proc. IEEE 13th Int. Conf. Softw. Testing, Validation Verification (ICST)*, 2020, pp. 375–386.

[27] M. Althoff and S. Lutz, "Automatic generation of safety-critical test scenarios for collision avoidance of road vehicles," in *Proc. IEEE Intell. Veh. Symp. (IV)*, 2018, pp. 1326–1333.

[28] R. Ben Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing advanced driver assistance systems using multi-objective search and neural networks," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2016, pp. 63–74.

[29] R. B. Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing vision-based control systems using learnable evolutionary algorithms," in *Proc. IEEE/ACM 40th Int. Conf. Softw. Eng. (ICSE)*. Piscataway, NJ, USA: IEEE, 2018, pp. 1016–1026.

[30] Z. Zhong, G. Kaiser, and B. Ray, "Neural network guided evolutionary fuzzing for finding traffic violations of autonomous vehicles," *IEEE Trans. Softw. Eng.*, vol. 49, no. 4, pp. 1860–1875, Apr. 2023.

[31] A. Gambi, T. Huynh, and G. Fraser, "Generating effective test cases for self-driving cars from police reports," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 257–267.

[32] "Avoiding carsickness when the cars drive themselves," *New York Times*, Jan. 2020. [Online]. Available: https://www.nytimes.com/2020/01/17/business/motion-sickness-self-driving-cars.html

[33] "Measuring motion sickness in driverless cars," Univ. of Michigan, Ann Arbor, MI, USA, Aug. 2019. [Online]. Available: https://news.umich.edu/measuring-motion-sickness-in-driverless-cars/

[34] A. K. Leung and K. L. Hon, "Motion sickness: An overview," *Drugs Context*, vol. 8, 2019.

[35] "Dreamland's grading system." GitHub. Accessed: May, 2023. [Online]. Available: https://bit.ly/3nfe48e

[36] On-Road Automated Vehicle Standards Committee et al., "Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems," *SAE Standard J.*, vol. 3016, pp. 1–16, 2014.

[37] Automated Vehicle Standards Committee, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," Jun. 2018, doi: 10.4271/J3016_201806.

[38] "Self-driving fundamentals: Featuring Apollo." Udacity. Accessed: May, 2023. [Online]. Available: https://www.udacity.com/course/self-driving-car-fundamentals-featuring-apollo--ud0419

[39] "Baidu Apollo: An open autonomous driving platform." Apollo. Accessed: May, 2023. [Online]. Available: http://apollo.auto/

[40] "Open vehicles compatible with Apollo." Apollo. Accessed: May, 2023. [Online]. Available: https://apollo.auto/vehicle/certificate_en.html

[41] R. Liao, "Baidu hits the gas on autonomous vehicles with Volvo and ford deals," *Tech Crunch*, Nov. 2018. [Online]. Available: https://techcrunch.com/2018/11/01/baidu-volvo-ford-autonomous-driving

[42] T. Kilgore, "Baidu debuts Robotaxi ride hailing service in China, using self-driving electric taxis," *Market Watch*, Sep. 2019. [Online]. Available: https://tinyurl.com/4yehrwxw

[43] "Anthropometric reference data for children and adults: United States, 2015–2018," Centers for Disease Control and Prevention, National Center for Health Statistics, Hyattsville, MD, USA, Jan. 2021. [Online]. Available: https://www.cdc.gov/nchs/data/series/sr_03/sr03-046-508.pdf

[44] "Bicycle road safety audit guidelines and prompt lists," Federal Highway Admin., U.S. Dept. Transp., Washington, DC, USA, May 2012. [Online]. Available: https://safety.fhwa.dot.gov/ped_bike/tools_solve/fhwasa12018/fhwasa12018.pdf

[45] "Federal size regulations for commercial motor vehicles," Federal Highway Admin., U.S. Dept. Transp., Washington, DC, USA, Jun. 1992. [Online]. Available: https://ops.fhwa.dot.gov/freight/publications/size_regs_final_rpt/size_regs_final_rpt.pdf

[46] "Analysis of lane-change crashes and near-crashes," Federal Highway Admin., U.S. Dept. Transp., Washington, DC, USA, Jun. 2009. [Online]. Available: https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/811147.pdf

[47] "ScenoRITA for Baidu Apollo v7.0.0." Zenodo. Accessed: May, 2023. [Online]. Available: https://doi.org/10.5281/zenodo.8231346

[48] A. Corso, P. Du, K. Driggs-Campbell, and M. J. Kochenderfer, "Adaptive stress testing with reward augmentation for autonomous vehicle validation," in *Proc. IEEE Intell. Transp. Syst. Conf. (ITSC)*, 2019, pp. 163–168.

[49] F. Hauer, A. Pretschner, and B. Holzmüller, "Fitness functions for testing automated and autonomous driving systems," in *Proc. Int. Conf. Comput. Saf., Rel., Secur.*, 2019, pp. 69–84.

[50] F. Kluck, M. Zimmermann, F. Wotawa, and M. Nica, "Genetic algorithm-based test parameter optimization for ADAS system testing," in *Proc. IEEE 19th Int. Conf. Softw. Qual., Rel. Secur. (QRS)*, 2019, pp. 418–425.

[51] H. Zhou et al., "Deepbillboard: Systematic physical-world testing of autonomous driving systems," in *Proc. IEEE/ACM 42nd Int. Conf. Softw. Eng. (ICSE)*, 2020, pp. 347–358.

[52] A. Stocco, M. Weiss, M. Calzana, and P. Tonella, "Misbehaviour prediction for autonomous driving systems," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, 2020, pp. 359–371.

[53] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems," in *Proc. 33rd IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*. Piscataway, NJ, USA: IEEE, 2018, pp. 132–142.

[54] J. Kim, J. Ju, R. Feldt, and S. Yoo, "Reducing DNN labelling cost using surprise adequacy: An industrial case study for autonomous driving," in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2020, pp. 1466–1476.

[55] Z. Peng, J. Yang, T.-H. Chen, and L. Ma, "A first look at the integration of machine learning models in complex autonomous driving systems: A case study on Apollo," in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2020, pp. 1240–1250.

[56] F. U. Haq, D. Shin, and L. Briand, "Efficient online testing for DNN-enabled systems using surrogate-assisted and many-objective optimization," in *Proc. IEEE/ACM 44th Int. Conf. Softw. Eng. (ICSE)*, 2022, pp. 811–822.

[57] J. Garcia, Y. Feng, J. Shen, S. Almanee, Y. Xia, and Q. A. Chen, "A comprehensive study of autonomous vehicle bugs," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, 2020, pp. 385–396.

[58] C. Erbsmehl, "Simulation of real crashes as a method for estimating the potential benefits of advanced safety technologies," in *Proc. 21st Int. Tech. Conf. Enhanced Saf. Veh. (ESV)*. Stuttgart, Germany: Nat. Highway Traffic Saf., 2009, pp. 9–162.

[59] "2021 IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST)," pp. 1–49, May 2021. [Online]. Available: https://sbst21.github.io/program/

[60] "2022 IEEE/ACM 15th International Workshop on Search-Based Software Testing (SBST)," pp. 1–55, May 2022. [Online]. Available: https://sbst22.github.io/program/

[61] F. Klück, L. Klampfl, and F. Wotawa, "Gabezier at the SBST 2021 tool competition," in *Proc. IEEE/ACM 14th Int. Workshop Search-Based Softw. Testing (SBST)*, 2021, pp. 38–39.

[62] E. Castellano, A. Cetinkaya, C. H. Thanh, S. Klikovits, X. Zhang, and P. Arcaini, "Frenetic at the SBST 2021 tool competition," in *Proc. IEEE/ACM 14th Int. Workshop Search-Based Softw. Testing (SBST)*, 2021, pp. 36–37.

[63] M. H. Moghadam, M. Borg, and S. J. Mousavirad, "Deeper at the SBST 2021 tool competition: ADAS testing using multi-objective search," in *Proc. IEEE/ACM 14th Int. Workshop Search-Based Softw. Testing (SBST)*, 2021, pp. 40–41.

[64] I. M. L. Rial and J. P. Galeotti, "EvoSuiteDSE at the SBST 2021 tool competition," in *Proc. IEEE/ACM 14th Int. Workshop Search-Based Softw. Testing (SBST)*, 2021, pp. 30–31.

[65] A. Abdullin, M. Akhin, and M. Belyaev, "Kex at the 2021 SBST tool competition," in *Proc. IEEE/ACM 14th Int. Workshop Search-Based Softw. Testing (SBST)*, 2021, pp. 32–33.

[66] L. Li, W.-L. Huang, Y. Liu, N.-N. Zheng, and F.-Y. Wang, "Intelligence testing for autonomous vehicles: A new approach," *IEEE Trans. Intell. Veh.*, vol. 1, no. 2, pp. 158–166, Jun. 2016.

[67] Y. Luo et al., "Targeting requirements violations of autonomous driving systems by dynamic evolutionary search," in *Proc. 36th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, 2021, pp. 279–291.

[68] H. Ebadi, M. Moghadam, M. Borg, G. Gay, A. Fontes, and K. Socha, "Efficient and effective generation of test cases for pedestrian detection - Search-based software testing of Baidu Apollo in SVL," in *Proc. IEEE Int. Conf. Artif. Intell. Testing (AITest)*. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, Aug. 2021, pp. 103–110, doi: 10.1109/AITEST52744.2021.00030.

[69] D. Kaufmann, L. Klampfl, F. Kluck, M. Zimmermann, and J. Tao, "Critical and challenging scenario generation based on automatic action behavior sequence optimization: 2021 IEEE autonomous driving AI test challenge group 108," in *Proc. IEEE Int. Conf. Artif. Intell. Testing (AITest)*. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, Aug. 2021, pp. 118–127, doi: 10.1109/AITEST52744.2021.00032.

[70] J. Seymour, D. Ho, and Q. Luu, "An empirical testing of autonomous vehicle simulator system for urban driving," in *Proc. IEEE Int. Conf. Artif. Intell. Testing (AITest)*. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, Aug. 2021, pp. 111–117, doi: 10.1109/AITEST52744.2021.00031.

[71] "The 2021 IEEE autonomous driving AI test challenge," in *Proc. IEEE Int. Conf. Artif. Intell. Testing (AITest)*," Jul. 2022. [Online]. Available: http://av-test-challenge.org/

[72] V. Nguyen, S. Huber, and A. Gambi, "Salvo: Automated generation of diversified tests for self-driving cars from existing maps," in *Proc. IEEE Int. Conf. Artif. Intell. Testing (AITest)*. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, Aug. 2021, pp. 128–135, doi: 10.1109/AITEST52744.2021.00033.

[73] K. Viswanadha et al., "Addressing the IEEE AV test challenge with Scenic and VerifAI," in *Proc. IEEE Int. Conf. Artif. Intell. Testing (AITest)*. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, Aug. 2021, pp. 136–142, doi: 10.1109/AITEST52744.2021.00034.

[74] R. V. Levine and A. Norenzayan, "The pace of life in 31 countries," *J. Cross-Cultural Psychol.*, vol. 30, no. 2, pp. 178–205, 1999.

[75] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," in in *Proc. Int. Conf. Parallel Problem Solving Nature*. Berlin, Heidelberg: Springer, 2000, pp. 849–858.

[76] S. Thrun et al., "The robot that won the DARPA Grand Challenge," *J. Field Robot.*, vol. 23, no. 9, pp. 661–692, Sep. 2006, doi: 10.1002/rob.20147.

[77] "PID speed control," Carnegie Mellon Robotics Academy, Pittsburgh, PA, USA, 2009. [Online]. Available: http://cmra.rec.ri.cmu.edu/previews/robot_c_products/teaching_rc_lego_v2_preview/reference/hp_PID.pdf

[78] "Who is at fault in a rear-end collision?" Ben Crump. Accessed: May, 2023. [Online]. Available: https://bencrump.com/car-accident-lawyer/who-is-at-fault-in-a-rear-end-collision/

[79] "Who is at fault in a car accident changing lanes in California." Bentley More. Accessed: May, 2023. [Online]. Available: https://www.bentleymore.com/who-is-at-fault-in-a-car-accident-changing-lanes/

[80] "Who is at fault in an accident when changing lanes?" Silkman Law Firm Injury. Accessed: May, 2023. [Online]. Available: https://silkmanlawfirm.com/blog/who-is-at-fault-in-an-accident-when-changing-lanes/

[81] P. Ingrassia, "Look, no hands! Test driving a Google car," *Reuters*, Aug. 2014. [Online]. Available: https://tinyurl.com/y65p4ufe

[82] H. Bellem, B. Thiel, M. Schrauf, and J. F. Krems, "Comfort in automated driving: An analysis of preferences for different automated driving styles and their dependence on personality traits," *Transp. Res. F, Traffic Psychol. Behav.*, vol. 55, pp. 90–100, 2018.

[83] M. Ester et al., "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. KDD*, 1996, vol. 96, no. 34, pp. 226–231.

[84] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[85] F. Hauer, I. Gerostathopoulos, T. Schmidt, and A. Pretschner, "Clustering traffic scenarios using mental models as little as possible," in *Proc. IEEE Intell. Veh. Symp. (IV)*, 2020, pp. 1007–1012.

[86] Q. Lin, W. Wang, Y. Zhang, and J. M. Dolan, "Measuring similarity of interactive driving behaviors using matrix profile," in *Proc. Amer. Contr. Conf. (ACC)*, 2020, pp. 3965–3970.

[87] W. Wang, A. Ramesh, J. Zhu, J. Li, and D. Zhao, "Clustering of driving encounter scenarios using connected vehicle trajectories," *IEEE Trans. Intell. Veh.*, vol. 5, no. 3, pp. 485–496, Sep. 2020.

[88] F. Kruber, J. Wurst, and M. Botsch, "An unsupervised random forest clustering technique for automatic traffic scenario categorization," in *Proc. 21st Int. Conf. Intell. Transp. Syst. (ITSC)*. Piscataway, NJ, USA: IEEE, 2018, pp. 2811–2818.

[89] J. Zhao, J. Fang, Z. Ye, and L. Zhang, "Large scale autonomous driving scenarios clustering with self-supervised feature extraction," 2021, *arXiv:2103.16101*.

[90] T. Zohdinasab, V. Riccio, A. Gambi, and P. Tonella, "DeepHyperion: Exploring the feature space of deep learning-based systems through illumination search," in *Proc. 30th ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, *ISSTA*. New York, NY, USA: ACM, 2021, pp. 79–90, doi: 10.1145/3460319.3464811.

[91] "Release apollo-v7.0.0 ApolloAuto/apollo." GitHub. Accessed: May, 2023. [Online]. Available: https://github.com/ApolloAuto/apollo/releases/tag/v7.0.0

[92] "Apollo's sim control." GitHub. Accessed: May, 2023. [Online]. Available: https://github.com/ApolloAuto/apollo/blob/266afbf68d83fa6fac7a812ff8a950223f5ab2c0/docs/FAQs/Dreamview_FAQs.md#whats-the-function-of-sim_control-in-the-backend-of-dreamview

[93] "Apollo 7.0 + LGSVL 2021.2 sometimes does not complete a sharp turn." GitHub. Accessed: May, 2023. [Online]. Available: https://github.com/ApolloAuto/apollo/issues/14756#issuecomment-1405921603

[94] K. Korosec, "Autonomous vehicle startup autox lands driverless testing permit in California," *Tech Crunch*, Feb. 2022. [Online]. Available: https://tinyurl.com/yscwmf9p

[95] "Study: The most dangerous intersections in Orange County." Aitken Aitken Cohn. Accessed: May, 2023. [Online]. Available: https://www.aitkenlaw.com/the-most-dangerous-intersections-in-orange-county/

[96] D. Shepardson, "U.S. agency to review if Pony.ai complied with crash reporting order," *Reuters*, Jul. 2022. [Online]. Available: https://tinyurl.com/42twsa2b

[97] "LGSVL Simulator." Accessed: May, 2023. [Online]. Available: https://www.lgsvlsimulator.com/

[98] "SVL Simulator Sunset." SVL Simulator. Accessed: May, 2023. [Online]. Available: https://www.svlsimulator.com/news/2022-01-20-svl-simulator-sunset/

[99] "Yuqihuai/SORA-SVL: Local SVL cloud." GitHub. Accessed: May, 2023. [Online]. Available: https://github.com/YuqiHuai/SORA-SVL

[100] "Cclinus/AV-Fuzzer." GitHub. Accessed: May, 2023. [Online]. Available: https://github.com/cclinus/AV-Fuzzer

[101] "Reimplementation-AVFuzzer." Zenodo. Accessed: May, 2023. [Online]. Available: https://zenodo.org/record/7898326#.ZFSk9RzMJaZ

[102] "AIasd/ADFuzz: An open-source software package for fuzzing autonomous driving systems in high-fidelity simulators." GitHub. Accessed: May, 2023. [Online]. Available: https://github.com/AIasd/ADFuzz

[103] A. Arcuri and G. Fraser, "On parameter tuning in search based software engineering," in *Search Based Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 33–47.

[104] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proc. 33rd Int. Conf. Softw. Eng.*, New York, NY, USA: IEEE, 2011, pp. 1–10.

[105] F.-A. Fortin, F.-M. De Rainville, M.-A. G. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 2171–2175, 2012.

[106] "Adfuzz/stack2_svl_apollo.md at main · AIasd/ADFuzz · GitHub." GitHub. Accessed: May, 2023. [Online]. Available: https://github.com/AIasd/ADFuzz/blob/main/doc/stack2_svl_apollo.md

[107] "Reproducing result for the archived paper #15." GitHub. Accessed: May, 2023. [Online]. Available: https://github.com/AIasd/ADFuzz/issues/15#issuecomment-1079088700

[108] "Error after 100 simulations." GitHub. Accessed: May, 2023. [Online]. Available: https://github.com/AIasd/ADFuzz/issues/11

[109] "Running ADFuzz for more than 1 hour." GitHub. Accessed: May, 2023. [Online]. Available: https://github.com/AIasd/ADFuzz/issues/22

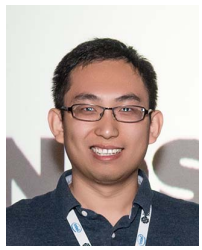
[110] A. Vargha and H. D. Delaney, "A critique and improvement of the CL common language effect size statistics of McGraw and Wong," *J. Educational Behav. Statist.*, vol. 25, no. 2, pp. 101–132, 2000.
[111] "ScenoRITA - Example scenario." Zenodo. Accessed: May, 2023. [Online]. Available: https://doi.org/10.5281/zenodo.8231392
[112] "Apollo-master camera object detection (Smoke)." GitHub. Accessed: May, 2023. [Online]. Available: https://github.com/lgsvl/simulator/issues/1719
[113] "Apollo-master LGSVL camera perception." GitHub. Accessed: May, 2023. [Online]. Available: https://github.com/ApolloAuto/apollo/issues/14072
[114] "3D Ground Truth Sensor can detect vehicles but not pedestrians." GitHub. Accessed: May, 2023. [Online]. Available: https://github.com/lgsvl/simulator/issues/1915
[115] "LGSVL pedestrian not showing up in Apollo." GitHub. Accessed: May, 2023. [Online]. Available: https://github.com/AIasd/ADFuzz/issues/6
[116] "3D ground truth sensor cannot detect pedestrian." GitHub. Accessed: May, 2023. [Online]. Available: https://github.com/lgsvl/simulator/issues/2137
[117] "Fix: Added missing predecessor and successor for Borregas avenue HD map." GitHub. Accessed: May, 2023. [Online]. Available: https://github.com/ApolloAuto/apollo/pull/14531
[118] N. Rahmah and I. S. Sitanggang, "Determination of optimal epsilon (EPS) value on DBSCAN algorithm to clustering data on peatland hotspots in Sumatra," in *Proc. IOP Conf. Ser., Earth Environ. Sci.*, Bristol, England: IOP Publishing, 2016, vol. 31, no. 1, Art. no. 012012.
[119] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach," *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 257–271, Nov. 1999.
[120] "Autoware: Open-source software for urban autonomous driving." GitHub. Accessed: May, 2023. [Online]. Available: https://github.com/CPFL/Autoware
[121] "Ecosystem," Autoware, Aug. 2021. [Online]. Available: https://autoware.org/ecosystem/
[122] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, Nov.–Dec. 2015.
[123] S. Kato et al., "Autoware on board: Enabling autonomous vehicles with embedded systems," in *Proc. ACM/IEEE 9th Int. Conf. Cyber-Physical Syst. (ICCPS)*. Piscataway, NJ, USA: IEEE, 2018, pp. 287–296.

Yuqi Huai received the B.S. degree in computer science and in software engineering from University of California, Irvine (UCI). He is working toward the Ph.D. degree in software engineering program with the Donald Bren School of Information and Computer Sciences at the UCI. He conducts research in software engineering with a focus on software architecture and software testing.

Sumaya Almanee received the M.Sc. degree in computer science from George Washington University and the Ph.D. degree in software engineering from the Donald Bren School of Information and Computer Sciences at the University of California, Irvine. She works on problems related to software security, testing, and analysis in smart systems.

Yuntianyi Chen received the B.S. degree in computer science from Wuhan University. He is working toward the Ph.D. degree in software engineering program with Donald Bren School of Information and Computer Sciences at the University of California, Irvine (UCI). He conducts research in software engineering with a focus on software configuration and software testing.

Xiafa Wu received the B.S. degree in computer science from University of California, Irvine (UCI). He is working toward the Ph.D. degree in computer science program with Donald Bren School of Information and Computer Sciences at the UCI. His current research interests include compilation, parallel software, and software analysis and security.



Qi Alfred Chen received the Ph.D. degree in computer science and engineering from the University of Michigan, Ann Arbor. He is an Assistant Professor with the School of Information and Computer Sciences at the University of California, Irvine. His research interests include software security, network protocol security, DNS system security, smartphone system security, CPS/IoT system security, access control system security, vulnerability discovery and analysis, and side channel attack and defense.

Joshua Garcia (Member, IEEE) received the B.S. degree in computer engineering and computer science from the University of Southern California (USC), and the M.S. and Ph.D. degrees in computer science from USC. He is an Assistant Professor with the School of Information and Computer Sciences at the University of California, Irvine. He conducts research in software engineering with a focus on software analysis and testing, software security, and software architecture. He is a member of the ACM and ACM SIGSOFT.