AdvFuzz: Finding More Violations Caused by the EGO Vehicle in Simulation Testing by Adversarial NPC Vehicles

ANONYMOUS AUTHOR(S)

Recently, there has been a significant escalation in both academic and industrial commitment towards the development of autonomous driving systems (ADSs). A number of simulation testing approaches have been proposed to generate diverse driving scenarios for ADS testing. However, scenarios generated by these previous approaches are static and lack interactions between the EGO vehicle and the NPC vehicles, resulting in a large amount of time on average to find violation scenarios. Besides, a large number of the violations they found are caused by aggressive behaviors of NPC vehicles, revealing no bugs of ADSs.

In this work, we propose the concept of adversarial NPC vehicles and introduce AdvFuzz, a novel simulation testing approach, to generate adversarial scenarios on main lanes (e.g., urban roads and highways). AdvFuzz allows NPC vehicles to dynamically interact with the EGO vehicle and regulates the behaviors of NPC vehicles, finding more violation scenarios caused by the EGO vehicle more quickly. We compare AdvFuzz with a random approach and three state-of-the-art scenario-based testing approaches. Our experiments demonstrate that AdvFuzz can generate 198.34% more violation scenarios compared to the other four approaches in 12 hours and increase the proportion of violations caused by the EGO vehicle to 87.04%, which is more than 7 times that of other approaches. Additionally, AdvFuzz is at least 92.21% faster in finding one violation caused by the EGO vehicle than that of the other approaches.

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

Additional Key Words and Phrases: Autonomous Driving System, Scenario-based Testing

ACM Reference Format:
Anonymous Author(s). 2018. AdvFuzz: Finding More Violations Caused by the EGO Vehicle in Simulation Testing by Adversarial NPC Vehicles. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 21 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

In recent decades, there has been a ground-breaking evolution in autonomous driving systems (ADSs). These systems represent the potential to enhance road safety, reduce traffic congestion, and improve transportation efficiency, revolutionizing the automotive transportation [47]. However, despite the advancements made by leading companies such as Tesla, Waymo, and Uber, current ADSs still struggle with corner cases and exhibit erroneous behaviors due to the extremely complicated real-world driving environments. These flaws in ADSs can lead to serious consequences and substantial losses, as highlighted by numerous documented traffic incidents [5, 30, 43]. Consequently, extensive testing is needed to ensure the safety and reliability of ADSs.

Leading companies have employed on-road testing to evaluate the performance of ADSs. However, autonomous vehicles have to be driven more than 11 billion miles to demonstrate with 95%

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/XXXXXXX.XXXXXXX

confidence that autonomous vehicles are 20% safer than human drivers [28]. This is not only time-consuming but also costly. In contrast, simulation testing offers a more efficient and cost-effective approach to generate diverse and challenging scenarios for ADSs by leveraging the high-fidelity simulators, such as LGSVL [34] and CARLA [15]. These simulators can generate a wide range of scenarios, including various weather conditions, road conditions, and traffic conditions.

Several simulation testing approaches have been proposed to generate critical scenarios for ADSs in simulator, such as DSL-based approaches [3, 4, 16, 48, 49], search-based approaches [1, 2, 8, 12, 20, 25, 26, 31, 35, 36, 39, 41, 51, 53, 54, 59, 60], and data-driven approaches [7, 13, 17, 57]. These approaches have been demonstrated to be capable of finding safety violations.

However, without the consideration of the EGO vehicle (i.e., the vehicle controlled by the ADS under test), these approaches often generate static scenarios where the behaviors of NPC vehicles are predefined, lacking the interactions between the EGO vehicle and NPC vehicles, inevitably suffering from the extensive and time-consuming exploration of the vast scenario search space. Besides, the NPC vehicles following the predefined behaviors may not obey traffic rules and collide with the EGO vehicle aggressively. As a result, violations found by these approaches do not necessarily reveal a bug in the ADS under test because the EGO vehicle may not bear the liability. This is also evidenced by a recent study [26], where 1,109 crash scenarios are automatically generated in 240 hours. After manual diagnosis, all these violations are caused by NPC vehicles.

To address these problems, we propose adversarial NPC vehicles that can adopt reasonable behaviors and dynamically adjust their maneuvers to improve the interactions with the EGO vehicle. We design and implement AdvFuzz, a novel simulation testing approach, to generate adversarial scenarios on main lanes (e.g., urban ways and highways) where NPC vehicles can interact with the EGO vehicle, finding more violation scenarios caused by the EGO vehicle more quickly. Specifically, we specify a segment of road as experimental field and implement the adversarial NPC vehicles by equipping the NPC vehicles in the simulator with the ability of adjusting their maneuvers guided by behavior trees [18] based on the real-time positions of the EGO vehicle. In addition, we adopt the genetic algorithm-based (GA-based) scenario generator and the scenario executor to support the automatic generation and correct execution of adversarial scenarios. After finding the violation scenarios (i.e., collision scenarios and rule-breaking scenarios), we use a rule-based liability determiner to diagnose the collision scenarios and exclude the violation scenarios caused by NPC vehicles.

We have conducted large-scale experiments to evaluate the effectiveness and efficiency of AdvFuzz. We implement AdvFuzz based on Apollo 8.0 [6] and LGSVL 2021.3 [34] and compare it with a random approach and three state-of-the-art scenario-based testing approaches (i.e., NSGAII-DT [1], AV-Fuzzer [36], and AutoFuzz [60]). Our experiments demonstrate that AdvFuzz can generate 198.34% more violations compared to other four approaches in 12 hours and increases the proportion of violations caused by the EGO vehicle to 87.04%. Besides, AdvFuzz is at least 51.98% faster in finding one violation scenario, 92.21% faster in finding one violation caused by the EGO vehicle, 58.32% faster in finding the first violation and 82.60% faster in finding the first violation caused by the EGO vehicle than those of the other approaches. Finally, we assess the effect of different configurations of the parameter in AdvFuzz on the effectiveness and efficiency results.

The main contributions of our work are summarized as follows:

- We propose adversarial NPC vehicles that can adopt reasonable behaviors and dynamically adjust their maneuvers to improve the interactions with the EGO vehicle guided by behavior trees.
- We design and implement a novel simulation testing approach, AdvFuzz, to automatically generate and execute adversarial scenarios in the simulator, maximizing the possibility of violation scenarios caused by the EGO vehicle.

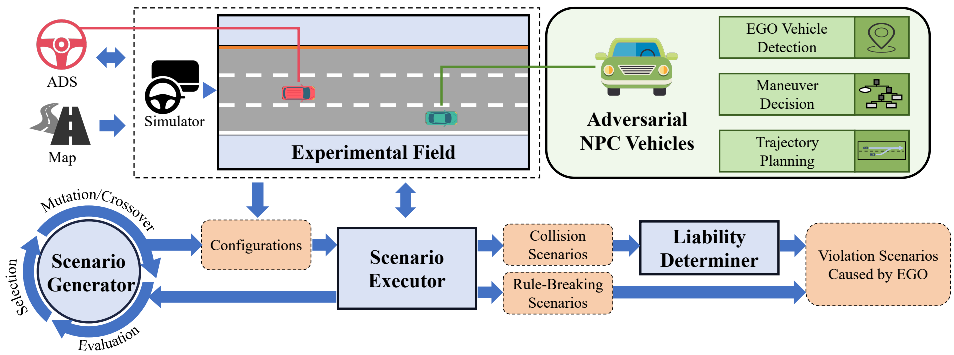


Fig. 1. Approach Overview of AdvFuzz

- We conduct experiments with a random approach and three state-of-the-art scenario-based testing approaches to demonstrate AdvFuzz's effectiveness and efficiency in finding violations caused by ADSs.

2 Methodology

We design and implement AdvFuzz to automatically generate adversarial scenarios on main lanes where NPC vehicles can interact with the EGO vehicle dynamically and find the violation scenarios caused by the EGO vehicle. The approach overview of AdvFuzz is presented in Fig. 1. The overall idea of AdvFuzz is to enhance the interaction between the EGO vehicle and NPC vehicles in simulation, finding more violation scenarios, and to regulate the behaviors of NPC vehicles, reducing the occurrence of violation scenarios caused by NPC vehicles. An adversarial scenario includes the ADS under test, an experimental field and several adversarial NPC vehicles.

We connect the ADS with the simulator and specify a segment of road on the map loaded in the simulator as the **Experimental Field** (see Sec. 2.1). To implement **Adversarial NPC Vehicles** (see Sec. 2.2), we equip the NPC vehicles in the simulator with three main functions: *EGO Vehicle Detection*, *Maneuver Decision*, and *Trajectory Planning*. These functions work together to enable adversarial NPC vehicles' interactions with the EGO vehicle, maximizing the possibility of safety-critical violations caused by the EGO vehicle. We randomly initialize a set of the configurations of adversarial scenarios and utilize a GA-based **Scenario Generator** (see Sec. 2.3) to automatically generate the adversarial scenarios. The **Scenario Executor** (see Sec. 2.4) is responsible for loading the experimental field as well as the configuration of adversarial scenarios, and executing scenarios with adversarial NPC vehicles. The violation scenarios found by the executor consist of collision scenarios and rule-breaking scenarios (i.e., the EGO vehicle breaks predefined rules). For all the collision scenarios, we utilize rule-based **Liability Determiner** (see Sec. 2.5) to eliminate the collision scenarios caused by adversarial NPC vehicles, and we thus get all the violation scenarios caused by the EGO vehicle at last.

2.1 Experimental Field

To support large-scale construction of adversarial scenarios, we select a segment of road provided by the map in the simulator as the experimental field. As shown in Fig. 2, there is an EGO vehicle (i.e., the red vehicle) controlled by ADS and several adversarial NPC vehicles (i.e., the green vehicles) in the experimental field. By default, the number of adversarial NPC vehicles is consistent with the number of lanes on the road. We define a "bubble" to manage adversarial NPC vehicles and limit the space where EGO vehicle and adversarial NPC vehicles can effectively and efficiently interact with each other. A bubble is a custom-defined region of length L (e.g., 300 meters) located a certain

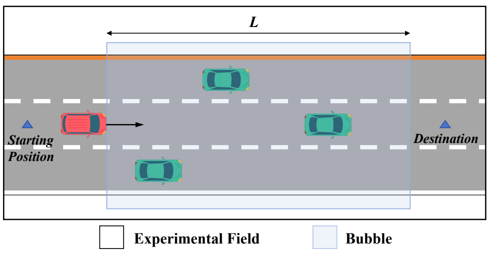


Fig. 2. The Architecture of Experimental Field

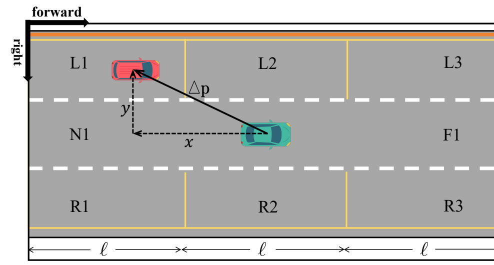


Fig. 3. Perception Zones of Adversarial NPC Vehicle

distance (e.g., 50 meters) in front of the starting position of EGO vehicle, within which adversarial NPC vehicles are randomly distributed. The task of the EGO vehicle is to pass through the bubble and reach the destination at the other end of the bubble. As the EGO vehicle enters the bubble with a stable speed, adversarial NPC vehicles in the bubble start to monitor the trajectory of EGO vehicle and make maneuvers trying to interact with ADS.

2.2 Adversarial NPC Vehicles

We introduce the implementation of *EGO Vehicle Detection* in Sec. 2.2.1, *Maneuver Decision* in Sec. 2.2.2, and *Trajectory Planning* in Sec. 2.2.3 for adversarial NPC vehicles.

2.2.1 EGO Vehicle Detection. Adversarial NPC vehicles are equipped with a function that enables them to locate the EGO vehicle. For each NPC vehicle in the bubble during the simulation, we define the perception zones around the NPC vehicle to determine the position relationship between the EGO vehicle and the NPC vehicle during driving. As shown in Fig. 3, when an adversarial NPC vehicle is driving in its lane, the adjacent left lane (if present) is divided into three zones: L1, L2, and L3. Similarly, the adjacent right lane (if present) is divided into R1, R2, and R3 zones. The area in front of the NPC vehicle is designated as F1, while the area behind it is labeled N1. Each of these zones, except for N1 and F1, has a length variable denoted as ℓ. The value of ℓ is set to 20 meters by default. The simulator provides the forward unit vector **forward** and the right unit vector **right**. We can also get the position p_E^t of the EGO vehicle and the position $p_{N_k}^t$ of the NPC vehicle N_k at any time t during the simulation. We use $\mathbf{p}_E^t$ and $\mathbf{p}_{N_k}^t$ to represent the vectors of p_E^t and $p_{N_k}^t$ respectively. The relative position between the two vehicles is $\triangle\mathbf{p} = \mathbf{p}_E^t - \mathbf{p}_{N_k}^t$. The values x and y respectively represent the projection lengths of $\triangle\mathbf{p}$ onto the **forward** and **right**. Assuming the road width is w, and given that x and y fall within the ranges $[-1.5\ell, 1.5\ell]$ and $[-1.5w, 1.5w]$ respectively if the EGO vehicle is located in the perception zones, the NPC vehicle can accurately locate the EGO vehicle's position within its perception zones as follows:

EGO is in N1 if $x < 0$ and $|y| \leq 0.5w$. EGO is in F1 if $x > 0$ and $|y| \leq 0.5w$.
EGO is in L1 if $x < -0.5\ell$ and $y < -0.5w$. EGO is in R1 if $x < -0.5\ell$ and $y > 0.5w$.
EGO is in L2 if $|x| \leq 0.5\ell$ and $y < -0.5w$. EGO is in R2 if $|x| \leq 0.5\ell$ and $y > 0.5w$.
EGO is in L3 if $x > 0.5\ell$ and $y < -0.5w$. EGO is in R3 if $x > 0.5\ell$ and $y > 0.5w$.

2.2.2 Maneuver Decision. For the purpose of enhancing the interactions between adversarial NPC vehicles and the EGO vehicle, while regulating NPC vehicles' behaviors, we use behavior trees [52] to make maneuver decisions. First, we define the maneuvers of an adversarial NPC vehicle as follows:

Definition 1. An adversarial NPC vehicle's maneuvers $\mathbb{M}$ in main lanes is a finite set of maneuvers including the following types:

- KEEP_SPEED is a maneuver to follow lane with a stable speed.
- ACCELERATION_STRAIGHT is a maneuver to speed up and go straight

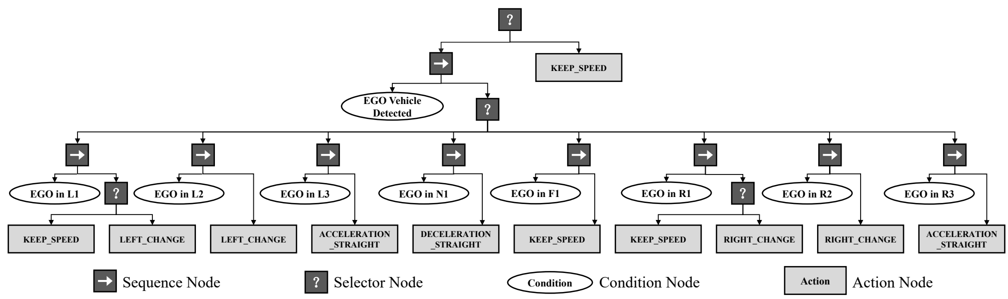


Fig. 4. The Graphical Representation of the Behavior Tree for Maneuver Decision.

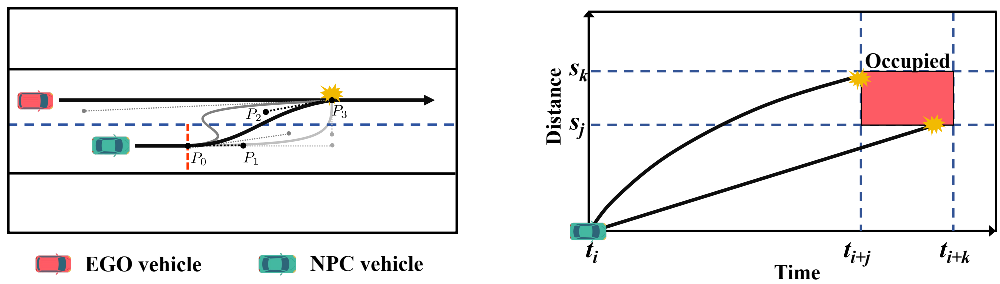
- DECELERATION_STRAIGHT is a maneuver to slow down and go straight.
- LEFT_CHNAGE is a maneuver to change lane to the left.
- RIGHT_CHANGE is a maneuver to change lane to the right.

An adversarial NPC vehicle can select a new maneuver $m \in \mathbb{M}$ to interact with the EGO vehicle after the previous one is completed during the simulation.

The behavior tree for adversarial NPC vehicles consists of four kinds of nodes. The *Sequence Node* composes actions or subtrees in an ordered fashion. The activation of its children is passed from one child to the next only if the current node is completed with a *SUCCESS* signal. Otherwise, a *FAILURE* signal is returned by the sequence node. The *Selector Node* composes actions or subtrees and activates its children in random order until one returns *SUCCESS*. A selector node returns *SUCCESS* if only one child node succeeds, otherwise it returns *FAILURE*. The *Condition Node* checks the position of the EGO vehicle and returns *SUCCESS* only if the condition is true. The *Action Node* executes specific maneuvers and returns *SUCCESS* when the maneuver is completed. The initial status of each action node (i.e., maneuver) is *IDLE*. When the NPC vehicle selects a maneuver, the status of the action node is set to *RUNNING*. Once the maneuver is completed, the action node returns *SUCCESS*, and the status of the action node transitions to *IDLE* (see Sec. 2.4).

Fig. 4 shows a graphical representation of our behavior tree for adversarial NPC vehicles in main lanes. Specifically, when the adversarial NPC vehicle fails to detect the EGO vehicle, it will move forward at a constant speed. Upon detecting the EGO vehicle, it can make adversarial maneuvers to interact with the EGO vehicle based on the zone where the EGO vehicle is located. If the EGO vehicle is in L1, the adversarial NPC vehicle may either continue driving straight at its current speed or change lanes to the left, with the choice influenced by a random seed. If the EGO vehicle is in L2, the adversarial NPC vehicle will change lanes to the left. We do not consider acceleration, deceleration, and right lane changes because the probability of these maneuvers leading to vehicle interactions is low at these conditions. If the EGO vehicle is in L3, the adversarial NPC vehicle will accelerate and drive straight until it passes the EGO vehicle. In this case, we hope that the adversarial NPC vehicle can accelerate to the front of the EGO vehicle to reduce the possibility of the NPC vehicle hitting the EGO vehicle from behind. Similarly, if the EGO vehicle is in N1, the adversarial NPC vehicle will decelerate to hinder the EGO vehicle. If the EGO vehicle is in F1, the adversarial NPC vehicle will maintain its original speed and continue driving straight. Besides, if the EGO vehicle is in R1, R2 or R3, the adversarial NPC vehicle's maneuver decisions can be analogous to those performed when the EGO vehicle is in L1, L2, or L3.

2.2.3 Trajectory Planning. After the maneuver decision is determined by the behavior tree at any time t_i during the simulation, the adversarial NPC vehicle generates a trajectory to execute the selected maneuver. Different trajectories of NPC vehicles will affect the EGO vehicle when executing the same maneuver. The trajectory planning process is to ensure that the adversarial



(a) Waypoints Generation of Adversarial NPC Vehicles (b) Speed Planning of Adversarial NPC Vehicles

Fig. 5. The Trajectory Planning for LEFT_CHANGE Maneuver

NPC vehicle effectively carries out the chosen maneuver to interact with the EGO vehicle. We define the trajectory of the adversarial NPC vehicle as follows:

Definition 2. A trajectory $T =< P^t_{N_k}, V^t_{N_k} >$ is a tuple where:

- $P^t_{N_k} =< p^0_{N_k}, p^1_{N_k}, \ldots, p^n_{N_k} >$ is a sequence of waypoints that the NPC vehicle is followed at each timestamp during the simulation. A waypoint p indicates a specific location on the map in the coordinate system. $p^0_{N_k}$ is the starting position, p^n_N is the end position.
- $V^t_{N_k} =< v^0_{N_k}, v^1_{N_k}, \ldots, v^n_{N_k} >$ is a sequence of speed of N_k at each timestamp during the simulation.

We divide trajectory planning process into two subtasks, namely ***Waypoints Generation*** and ***Speed Planning***. We take the trajectory planning for LEFT_CHANGE maneuver as an example, as illustrated in Fig. 5. Other maneuvers' implementation is available at our replication site due to space limitations.

Waypoints Generation. The Waypoints Generation process is responsible for generating a sequence of waypoints that the adversarial NPC vehicle should follow to execute the selected maneuver. As shown in Fig. 5(a), the adversarial NPC vehicle's waypoints are divided into two phases. The first phase is the straight driving phase in the original lane. This phase is crucial for setting up a natural and realistic maneuver, as it prevents the adversarial NPC vehicle from making a sudden or unexpected lane change. The second phase of waypoints is the lane change phase. The adversarial NPC vehicle transitions from its current lane to the target lane as determined by the maneuver decision. Bézier curves [44], which are parametric curves providing a smooth transition, are used to calculate the waypoints for this phase. A Bézier curve $B(t)$ can be constructed by four control points $P_0 - P_3$, i.e., $B(\zeta) = (1-\zeta)^3 P_0 + 3(1-\zeta)^2\zeta P_1 + 3(1-\zeta)\zeta^2 P_2 + \zeta^3 P_3,\ \zeta \in [0,1]$. We set the current position of the adversarial NPC vehicle as P_0, and the target lane position extracted from map as P_3. We randomly set the two additional points (i.e., P_1 and P_2) and exclude waypoint curves with direction inversion, lane departure and sharp turns, which may not occur in real world (e.g., gray waypoints in Fig. 5(a)).

Speed Planning. In order to adjust the adversarial NPC vehicle's speed maximizing the possibility of a collision with the EGO vehicle, we use the s-t graph [42] shown in Fig. 5(b) to plan the appropriate speed profile for the adversarial NPC vehicle to follow these waypoints generated by *Waypoints Generation*. Assuming that the total length of the waypoints generated by the adversarial NPC vehicle is s and the EGO vehicle will travel at a constant speed v^i_E since time t_i, then, the EGO vehicle will occupy the s_j to s_k section of the NPC vehicle's waypoints from time t_{i+j} to t_{i+k}. Therefore, there is an occupied area in the s-t graph where a collision is most likely to occur. We can generate s-t curves through this occupied area and calculate the change in slope of the curve to plan adversarial NPC vehicles' speeds that are prone to collision.

2.3 Scenario Generator

To automatically generate the adversarial scenarios, we use the NSGA-II (Non-dominated Sorting Genetic Algorithm II) [62] to generate the configurations for adversarial scenarios. An adversarial scenario configuration is defined as follows.

Definition 3. An adversarial scenario configuration $Conf =< E, \mathbb{N}, W, t^S >$ is a tuple where:

- $E =< p_E^0, p_{des} >$ is the EGO vehicle controlled ADS under test, consisting of the starting position p_E^0 and the destination p_{des} of the EGO vehicle.
- $\mathbb{N} = \{p_{N_0}^0, p_{N_1}^0, \ldots, p_{N_{|\mathbb{N}|-1}}^0\}$ is a finite set of adversarial NPC vehicles where $|\mathbb{N}| > 0$. Each adversarial NPC vehicle N_k is initialized by the starting positions $p_{N_k}^0$ in the bubble randomly. Note that, we do not need to configure the waypoints and speed of adversarial NPC vehicles because they are dynamically generated in Sec. 2.2.3.
- $W =< rain, fog, wetness, cloudness, time >$ is a tuple used to specify weather conditions and the time of the day. $rain$, fog, $wetness$ and $cloudness$ are float numbers ranging from 0 to 1 and $time$ is an integer between 0 and 24.
- t^S is the maximum allowed frames duration for the scenario. We divide 1 second into 10 frames and set t^S to 500 frames by default ensuring enough time for the EGO vehicle to go through the bubble.

An individual I in the population is an adversarial scenario configuration consisting of four chromosomes c_0-c_2 (i.e., E, $\mathbb{N}$ and W). Each chromosome c has at least one gene (e.g., p_E^0). We obtain the coordinate range of the entire experimental field and the bubble from the simulator to get the positions where the EGO vehicle and the adversarial NPC vehicle can be placed. Furthermore, we randomly initialize the first generation of adversarial scenario configurations.

Mutation. With equal probability, one of the three chromosomes (i.e., E, $\mathbb{N}$ and W) in I will be mutated to increase diversity in the population. Mutating a specific c implies one of the genes in c will be mutated randomly. For example, the p_{des} of EGO vehicle in c_0 may be changed to a new position or the $rain$ in c_2 may be changed to a new float.

Crossover. With equal probability, one of the three chromosomes (i.e., E, $\mathbb{N}$ and W) in I will be selected for crossover to produce offspring. We utilize single-point crossover and the crossover point is chosen randomly within c. For example, if the crossover point in c_1 is $p_{N_k}^0$, the $p_{N_k}^0 \ldots p_{N_{|\mathbb{N}|-1}}^0$ part in c_1 of I_0 will be exchanged with that in c_1 of I_1 to generate two offspring.

Fitness Evaluation. To search for adversarial scenarios that maximize the possibility of safety violations, we collect the record of adversarial scenarios after simulation and consider the following three objectives based on prior works [12, 21, 31] to evaluate the performance of the ADS.

(1) Reaching Destination. Given the waypoints $< p_E^0, p_E^1, \ldots, p_E^n >$ of EGO vehicle and its destination p_{des}. The distance of EGO vehicle to its destination after simulation should meet Eq. 1,

$$D_{E2des}(p_E^n, p_{des}) \leq threshold \tag{1}$$

where the $threshold$ is set to half of the length of the bounding box of the EGO vehicle in our work.

(2) Collision Avoidance. Given the waypoints $< p_A^0, p_A^1, \ldots, p_A^n >$ of each vehicle in the execution record of an adversarial scenario S. Let $D_c(E, S)$ denote the minimum distance between the EGO vehicle and the NPC vehicles in S, we can calculate it through Eq. 2,

$$D_c(E, S) = \min\left(\left\{D_{E2N}(p_E^t, p_{N_k}^t) \mid 0 \leq k < |\mathbb{N}|, 0 \leq t \leq t^S\right\}\right) \tag{2}$$

where $D_{E2N}(p_E^t, p_{N_k}^t)$ calculates the shortest distance between the bounding box of the EGO vehicle E and the NPC vehicle N_k at time t.

(3) Not Hitting Illegal Lines. Given the waypoints $< p_E^0, p_E^1, \ldots, p_E^n >$ of EGO vehicle and a set of illegal lines (e.g., yellow lines or edge lines) denoted as $Lines$ extracted from map, the minimum

distance $D_l(E,S)$ between the EGO vehicle and *Lines* during the execution is defined by Eq. 3,

$$D_l(E,S) = \min\left(\left\{D_{E2l}(p_E^t, l) \mid l \in Lines, 0 \leq t < t^S\right\}\right) \tag{3}$$

where $D_{E2l}(p_E^t, l)$ calculates the shortest distance between the bounding box of the EGO vehicle and the illegal line l at time t.

The fitness function F that combines these three objectives to evaluate the safety of the ADS is defined by Eq. 4,

$$F = (f_1, f_2, f_3) = \left(D_{E2des}(p_E^n, p_{des}), \frac{w_1}{D_c(E,S)}, \frac{w_2}{D_l(E,S)}\right) \tag{4}$$

where w_1 and w_2 are constants. The larger $D_{E2des}(p_E^n, p_{des})$, the smaller $D_c(E,S)$ and the smaller $D_l(E,S)$ indicate that the scenario is more likely to have a violation.

Selection. First, we combine the current generation G_t with its offspring Q_t generated by mutation and crossover to form a new population R_t. Then, we sort the individuals in R_t based on a Pareto order [62]. After sorting, each individual will be assigned a crowding distance, which measures the proximity of individuals to each other. A larger average crowding distance enhances population diversity. Selection favors individuals with lower ranks in the Pareto order, and in cases of ties, favors those with greater crowding distances. Only the best τ individuals are selected to construct the next generation G_{t+1}, where τ is the population size of each generation.

Restart. We also employ a random restart mechanism similar to the previous work [36, 53] to resolve the convergence issue of genetic algorithm, which will be triggered by stagnation of five consecutive generations, that is, the individuals in five consecutive generations fail to achieve a higher fitness value. When the restart mechanism is triggered, the population will be reinitialized. The restart mechanism will also record the chromosomes of adversarial scenario configurations that have been generated and avoid generating the same configurations again.

2.4 Scenario Executor

When we obtain the configuration of an adversarial scenario from Scenario Generator, the Scenario Executor will utilize adversarial NPC vehicles implemented in Sec. 2.2 to execute the scenario and record simulation results. The process of the scenario execution is represented in Algorithm 1. The input of Scenario Executor includes the experimental field *Field* and the configuration of adversarial scenarios $Conf$. The output of Scenario Executor is the record of the simulation *Record*.

First, we initialize the adversarial scenario (Line 1-5). Specifically, we instantiate the adversarial NPC vehicles according to $Conf.\mathbb{N}$ (Line 1). We load the experimental field *Field* as well as adversarial NPC vehicles into the simulator *sim*, and we initialize *sim* using E and W in $Conf$, bridging it with ADS (Line 2). For each adversarial NPC vehicle, we set all its maneuvers'.

Second, we run the simulation loop for a total simulation frames of t^S configured in $Conf$ (Line 6-17). Each frame represents a simulation time step of 0.1 seconds (Line 16). For each NPC vehicle, if all its maneuvers' statuses are *IDLE*, it will detect the zone where the EGO vehicle is located (Line 9) and decide the maneuver m based on the zone (Line 10). Then it will plan the trajectory of m (Line 11) and set the status of m to *RUNNING* (Line 12). Besides, we start an asynchronous task to execute and monitor m by calling the function *Monitor_signal* (Line 13). When the simulation is completed, we update the simulation record (Line 18) and return it for evaluation (Line 19).

For the function *Monitor_signal*, it is an asynchronous function independent of the simulation loop (Line 20-23). We execute the maneuver m and wait for the *SUCCESS* signal of m (Line 21). Then we set the status of m to *IDLE* (Line 22).

Scenario Executor will also monitor whether the EGO vehicle violates the rules (i.e., reaching destination and not hitting illegal lines) during the simulation. Note that, when a vehicle collision

Algorithm 1: Execution of an Adversarial Scenario

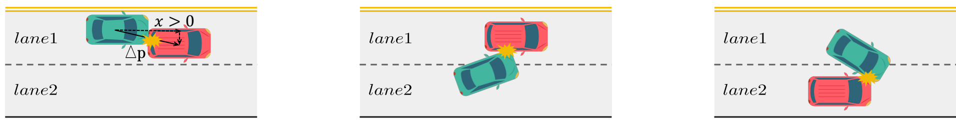
```
Input: the experimental field: Field, the configuration of adversarial scenarios: Conf
Output: the record of the simulation: Record
NPC_list ← Init_AdvNPC(Conf.ℕ);
sim ← Initialize(Field, Conf.E, Conf.W, NPC_list);
foreach NPC ∈ NPC_list do
    Set the status of all maneuvers m ∈ NPC.𝕄 to IDLE;
end
for t ← 1 to Conf.t^S do
    foreach NPC ∈ NPC_list do
        if all maneuvers m ∈ NPC.𝕄 have status IDLE then
            zone ← NPC.Detect_ego(sim.EGO);
            m ← NPC.Decide_maneuver(zone);
            m.trajectory ← NPC.Plan_trajectory(m, sim.EGO);
            m.status ← RUNNING;
            Monitor_signal(m);
        end
    end
    sim.run(0.1);
end
Record ← Update_record(sim, EGO, NPC_list);
return Record;
// Asynchronously monitor the execution status of maneuver.
Function Monitor_signal(m):
    if m.execute() = SUCCESS then
        m.status ← IDLE;
    end
```

occurs and is detected by the simulator's callback function, the simulation will end immediately and return the simulation record before the collision.

2.5 Liability Determiner

We denote violation scenarios caused by the EGO vehicle as EGO_Fault and violation scenarios caused by NPC vehicles as NPC_Fault. In a collision scenario, we need to determine liability and distinguish between EGO_Fault and NPC_Fault. According to the Uniform Vehicle Code (UVC) [46], the rear vehicle is generally responsible in rear-end collisions, while lane changers are liable if a collision occurs during a lane change. Specifically, if vehicle-A attempts a lane change but fails to complete it, resulting in a collision with the vehicle-B, it is considered to be caused by vehicle-A. Conversely, if the vehicle-A successfully switches lanes and the vehicle-B collides with its rear, it is considered to be caused by vehicle-B.

For all collisions identified by AdvFuzz, liability is determined based on the relative position of the vehicles and the status of NPC vehicle's maneuver. Given the relative position $\triangle \mathbf{p} = \mathbf{p}_E^t - \mathbf{p}_{N_k}^t$ at time t between the EGO vehicle E and the NPC vehicle N_k, the value x calculated by the projection of $\triangle \mathbf{p}$ onto forward unit vector **forward** indicates the front and rear relationship of vehicles. We intercept the EGO vehicle's trajectory within 30 frames before the collision and match it with the lane coordinates in the map to determine whether the EGO vehicle has crossed lines. If the EGO vehicle has crossed lanes, we denote it as $Switched(E) = True$, otherwise, $Switched(E) = False$. We also can obtain the lanes l_E and l_{N_k} where E and N_k are located in when the collision occurs according to their bounding boxes in the map respectively. For example, if the bounding box of the EGO vehicle is entirely in $lane1$, we have $l_E = lane1$.



(a) Situation 1 (b) Situation 2 (c) Situation 3

Fig. 6. Examples of Violations Caused by Adversarial NPC Vehicle

As shown in Fig. 6, we use a red vehicle to represent the EGO vehicle and a green vehicle to represent the adversarial NPC vehicle. If $l_E = l_{N_k}$ and $x > 0$, i.e., the NPC vehicle rear-ends the EGO vehicle (see Fig. 6(a)), we consider the collision as NPC_Fault. Furthermore, if the NPC vehicle is performing a LEFT_CHANGE or RIGHT_CHANGE maneuver with a status of $RUNNING$ and $Switched(E) = False$, i.e., the NPC vehicle collides with the EGO vehicle that is in the adjacent lane and has lane rights when changing lanes (see Fig. 6(b)) or the EGO vehicle slows down but can not avoid a collision with the NPC vehicle that changes lanes aggressively (see Fig. 6(c)), we also consider these collision scenarios as NPC_Fault.

The Liability Determiner eliminates the NPC_Fault in the collision scenarios and combine the remaining collision scenarios with rule-breaking scenarios, obtaining the violations caused by the EGO vehicle.

3 Evaluation

To evaluate the effectiveness and efficiency of AdvFuzz in generating diverse safety violations, we design the following three research questions.

- **RQ1 Effectiveness Evaluation**: How effective is AdvFuzz in finding safety violations compared to other approaches?
- **RQ2 Efficiency Evaluation**: How efficient is AdvFuzz in finding safety violations compared to other approaches?
- **RQ3 Parameter Sensitivity Evaluation**: How does the parameter ℓ of adversarial NPC vehicle's perception zones affect the AdvFuzz's performance?

3.1 Evaluation Setup

Target ADS and Simulation Platform. We choose Baidu Apollo [6] as our target ADS, which is one of the most representative industrial-grade full-stack ADSs with widespread commercialization. Specifically, we use the latest stable version of Apollo (i.e., Apollo 8.0). We select LGSVL 2021.3 [34] as our simulation platform because LGVSL [50] offers stable connections with Apollo. Although the remote service of LGSVL is no longer maintained, we use a local version [24].

Prototype. We implement a prototype of AdvFuzz with 9,018 lines of Python code. Our prototype uses LGSVL Python APIs [33] for scenario execution and violation detection. During the process of simulation, Apollo 8.0 is equipped with a wide range of sensors, including two camera sensors, one GPS, one radar and one LiDAR. All modules of Apollo are turned on, including perception module, localization module, prediction module, routing module, planning module, control module. Besides, we choose the SanFrancisco map in the SVL map library which contains various types of roads.

Baselines. First, we compare AdvFuzz with a random approach denoted as Rand that generates scenarios with multiple NPC vehicles that travel at random speeds, ignoring traffic rules and other road participants. Additionally, we also compare AdvFuzz with three state-of-the-art open-source scenario-based testing approaches, i.e., NSGAII-DT [1], AV-Fuzzer [36] and AutoFuzz [60].

Research Question Setup. For **RQ1**, we run AdvFuzz and other four approaches for 12 hours generating adversarial scenarios respectively. We run AdvFuzz in a two-lane urban way and a

Table 1. Effectiveness of AdvFuzz Compared with Other Approaches

Metrics	2-lane Urban Way					4-lane Highway
	Rand	NSGAII-DT	AV-Fuzzer	AutoFuzz	AdvFuzz	AdvFuzz
Scenario_Num	900	922	566	917	708	681
Violation_Num	92	211	260	163	540	340
EGO_Fault_Num	5	13	37	28	470	233
Proportion	5.43%	6.16%	14.10%	16.92%	87.04%	68.53%
Types_Num	2	2	5	6	10	14 (10 + 4)

four-lane highway while other approaches are only run in the two-lane urban way. This is because the open-source versions of other approaches only provide scenario configurations for 2-lane urban ways and are difficult to migrate. For the other four approaches, given that they are not equipped with the Liability Determiner, two authors manually verify whether the violations are caused by EGO vehicle and the Cohen Kappa coefficient reaches 0.862. We evaluate the effectiveness of AdvFuzz from the following aspects: (1) How many scenarios can be generated in 12 hours? (2) How many violations can be detected in 12 hours? (3) What is the number of EGO_Fault and its proportion among all the violations? (4) How many types of EGO_Fault can be found? (5) How are the speed changes of the NPC vehicles during the simulation?

For **RQ2**, we compare AdvFuzz with other four approaches from the following five aspects: (1) How much time does it take to generate one scenario on average? (2) How much time does it take to find one violation on average? (3) How much time does it take to find one EGO_Fault on average? (4) How much time does it take to find the first violation scenario? (5) How much time does it take to find the first violation scenario caused by EGO vehicle?

For **RQ3**, we set the parameter ℓ of adversarial NPC vehicle's perception zones as 20, 30, and 40 meters, denoted as AdvFuzz-20, AdvFuzz-30 and AdvFuzz-40 respectively. We generate scenarios in the two-lane urban way and the four-lane highway for 12 hours and record the results.

We run all the above experiments 3 times, and report the average results.

Experiment Environment. We conduct all the experiments on an Ubuntu 22.04.4 LTS server with an NVIDIA GeForce RTX 3090 GPU, Intel Core i9-13900K (32) CPU with 5.500GHz processor and 64GB memory.

3.2 Effectiveness Evaluation (RQ1)

Overall Results. Table 1 presents the effectiveness of AdvFuzz in generating safety violations compared to other approaches. In the two-lane urban way, with respect to the number of scenarios generated, AdvFuzz generates 708 scenarios in 12 hours. NSGAII-DT generates the most scenarios up to 922, while AV-Fuzzer generates 566 scenarios, which is the fewest among the five approaches. This shows that although AdvFuzz takes time to dynamically calculate the behaviors of NPC vehicles, it is not significantly slower than other approaches in generating scenarios. With respect to the number of violations detected in 12 hours, AdvFuzz finds 540 violations, while other four approaches can only find 181 violations on average. With respect to the proportion of violations caused by the EGO vehicle, AdvFuzz detects 470 violations caused by the EGO vehicle accounting for 87.04% of total while no more than 20% of the violations detected by other approaches are caused by the EGO vehicle. The proportion of violation scenarios caused by the EGO vehicle found by our tool increases by 717.09% on average compared to other four approaches. With respect to the types of violations caused by EGO vehicle, AdvFuzz finds 10 types of violations. Rand and NSGAII-DT find 2 types of violations (i.e., example 1 and 2 of AdvFuzz). AV-Fuzzer finds 5 types of violations (i.e., example 1, 2, 3, 5 and 9 of AdvFuzz) and AutoFuzz finds 6 types of violations (i.e., example 1, 2, 3, 5, 9 and 10 of AdvFuzz).

(a) Example of Type 1 (b) Example of Type 2
(c) Example of Type 3 (d) Example of Type 4
(e) Example of Type 5 (f) Example of Type 6
(g) Example of Type 7 (h) Example of Type 8
(i) Example of Type 9 (j) Example of Type 10
(k) Example of Type 11 (l) Example of Type 12
(m) Example of Type 13 (n) Example of Type 14

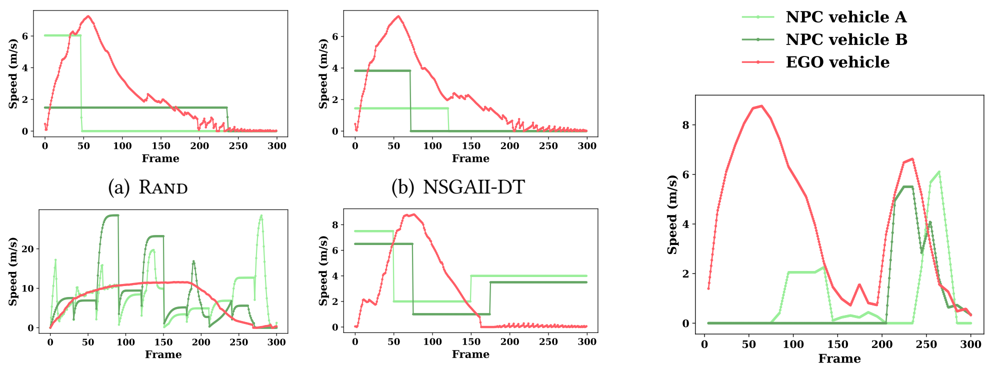
Fig. 7. Examples of Violations Caused by EGO Vehicle

In the four-lane highway, AdvFuzz also shows strong performance that it generates 681 scenarios in 12 hours and detects 340 violations in total. 68.53% of the violations (i.e., 233 violations) are caused by the EGO vehicle. Furthermore, in addition to the 10 types of violations found in the 2-lane urban way, AdvFuzz also finds 4 additional violation scenarios caused by the EGO vehicle (i.e., example 11, 12, 13 and 14).

Examples of Violation Types. We show an example for each type of violations caused by EGO vehicle found by AdvFuzz in the two-lane urban way and the four-lane highway. We use a red vehicle to represent the EGO vehicle and green vehicles to represent the adversarial NPC vehicles.

- **Example of Type 1:** as depicted in Fig. 7(a), the EGO vehicle rear-ends an NPC vehicle that is changing lanes. In this scenario, the EGO vehicle is driving along the lane, and the NPC vehicle successfully changes lanes in front of the EGO vehicle from a short distance ahead in the adjacent lane. However, the EGO vehicle fails to decelerate in time and rear-ends the NPC vehicle. This violation occurs because the prediction module fails to accurately anticipate the lane-changing intention of the NPC vehicle. Consequently, the EGO vehicle fails to decelerate in time and maintain a safe distance, leading to a rear-end collision.
- **Example of Type 2:** as depicted in Fig. 7(b), the EGO vehicle hits the side of an NPC vehicle that attempts to change lanes. In this scenario, the EGO vehicle attempts to change lanes to the adjacent lane to reach the destination, but the NPC vehicle in front of EGO vehicle nearly changes lanes to the EGO vehicle's lane. Finally, the EGO vehicle hits the rear side of the NPC vehicle. This violation occurs because the EGO vehicle is unable to accurately predict the intentions of other vehicles when changing lanes and fails to change lanes within a reasonable distance.

- **Example of Type 3:** as depicted in Fig. 7(c), the EGO vehicle collides with an NPC vehicle that finishes lane change. In this scenario, the NPC vehicle A initiates a right change maneuver while the EGO vehicle attempts to change to the left. Then, the NPC vehicle B also begins to change lanes to the adjacent lane. Finally, the EGO vehicle hits the NPC vehicle B that has completed the lane change. The EGO vehicle ignores the actions of the NPC vehicle B and fails to slow down in time and maintains a safe distance after changing lanes.
- **Example of Type 4:** as depicted in Fig. 7(d), the EGO vehicle hits the rear of an NPC vehicle. In this scenario, the EGO vehicle is driving along the lane, and the NPC vehicle A attempts to change lanes to the EGO vehicle's lane. Then, the EGO vehicle accelerates forward. At this time, the NPC vehicle B also begins to change lanes. Finally, the EGO vehicle collides with NPC vehicle B. This violation occurs because when the EGO vehicle accelerates to avoid the vehicle behind, it fails to maintain a safe distance from the NPC vehicle that changes lanes in front.
- **Example of Type 5:** as depicted in Fig. 7(e), the EGO vehicle hits other NPC vehicles stuck on lane. In this scenario, the EGO vehicle is driving along the lane following the NPC vehicle A. The NPC vehicle B attempts to change lanes to the EGO vehicle's lane, but collides with the NPC vehicle A. Then both of the NPC vehicles stop on the road. Finally, the EGO vehicle collides with the stationary NPC vehicle. The EGO vehicle interprets the two stationary NPC vehicles as one located in the adjacent lane, losing the perception result of NPC vehicle A. It is too late to slow down when it detects the NPC vehicle A again and ends up colliding with the NPC vehicle.
- **Example of Type 6:** as depicted in Fig. 7(f), the EGO vehicle changes lanes across a yellow line. In this scenario, the EGO vehicle follows the NPC vehicle A which moves at a slow speed. The EGO vehicle tries to overtake the NPC vehicle A, but the right lane is occupied by another NPC vehicle B. Then the EGO takes the action of crossing the yellow line to change lanes.
- **Example of Type 7:** as depicted in Fig. 7(g), there is a side collision between the EGO vehicle and the NPC while the EGO also hits the yellow line. In this scenario, the EGO vehicle tries to avoid the NPC vehicle that is changing lanes on the right, but it chooses to turn left and drives on the yellow line, and eventually collides with the NPC from the side.
- **Example of Type 8:** as depicted in Fig. 7(h), the EGO vehicle fails to plan trajectory and does not reach the destination. In this scenario, the EGO vehicle follows a slow NPC vehicle, and it tries to change lanes to the adjacent lane for overtaking. When the EGO vehicle is halfway through changing lanes (i.e. on the lane line), the NPC vehicle in front continues to drive and gives way for a distance. At this time, EGO begins to plan the trajectory again, wavering between changing lanes and continuing to follow the vehicle, and finally got stuck.
- **Example of Type 9:** as depicted in Fig. 7(i), the EGO vehicle hits the rear of an NPC vehicle. In this scenario, the NPC vehicle A rear-ends NPC vehicle B. The EGO vehicle, approaching the scene, detects the two NPC vehicles as one and fails to stop immediately and consequently rear-ends NPC vehicle A.
- **Example of Type 10:** as depicted in Fig. 7(j), the EGO vehicle side-collides with an NPC vehicle. In this scenario, the EGO vehicle attempts to overtake the NPC vehicle B, while the NPC vehicle A is driving on the adjacent lane. However, during the overtaking process, the EGO vehicle fails to adequately consider the position and speed of NPC vehicle A, resulting in a side collision with NPC vehicle A. The violation in this scenario can be attributed to the prediction and planning module of the EGO vehicle that fail to accurately predict the trajectory of NPC vehicle A and generate a safe trajectory for the EGO vehicle.
- **Example of Type 11:** as depicted in Fig. 7(k), the EGO vehicle hits the side of the NPC vehicle. In this scenario, the EGO vehicle and NPC vehicle both attempt to change into the same lane simultaneously, resulting in a side collision. The EGO vehicle fails to effectively detect and predict the NPC vehicle's approaching from the side. Insufficient side detection caused the planning



(a) Rand (b) NSGAII-DT (c) AV-Fuzzer (d) AutoFuzz (e) AdvFuzz

Fig. 8. Speed Changes of Vehicles in the 2-lane Urban Way

module to forcibly perform lane changes maneuver without considering the presence of NPC vehicles, ultimately leading to side collisions

- **Example of Type 12:** as depicted in Fig. 7(l), the EGO vehicle collides with NPC vehicle when changing lines. In this Scenario, the EGO vehicle performs a lane change to the left in response to NPC vehicle B's slow movement in the lane ahead. However, it ignores that NPC vehicle A is changing lanes to the right. Finally, the EGO vehicle hit the NPC vehicle A that almost completes lane change. The failure arises due to the EGO vehicle's inability to predict the NPC vehicle A's lane change and the failure to generate a safe trajectory.
- **Example of Type 13:** as depicted in Fig. 7(m), the EGO vehicle collides with two NPC vehicles. In this scenario, while the EGO vehicle is moving forward, both NPC vehicle A and NPC vehicle B attempt to change lanes into the EGO vehicle's lane and collide with each other, blocking the way of EGO vehicle. However, the EGO vehicle fails to recognize the collision between the two NPC vehicles and continues moving forward, ultimately resulting in a collision with the stationary NPC vehicles.
- **Example of Type 14:** as depicted in Fig. 7(n), the EGO vehicle fails to plan trajectory and reach the destination. In this scenario, the EGO vehicle is moving forward while NPC vehicle A attempts to change lanes. During this maneuver, NPC vehicle A collides with NPC vehicle B, causing both NPC vehicles to stop on the roadway. The EGO vehicle continues to move forward slowly until it reaches a point where the ADS determines it is unsafe to proceed further. At this point, the EGO vehicle is unable to plan a route around the stationary NPC vehicles even though there is enough space to change lanes and the lane is not occupied.

Speed Change Analysis. Besides, we record the speed changes of the vehicles during the simulation. Fig. 8 shows an example of the vehicles' speed changes across different testing approaches in the two-lane urban way respectively. We use green lines to represent the speed changes of NPC vehicles and red lines to represent the speed changes of the EGO vehicle. As shown in Fig. 8(a) and Fig. 8(b), the speed changes of the NPC vehicles in Rand and NSGAII-DT are relatively simple that the NPC vehicles usually travel a certain distance at a stable speed, then suddenly stop on the road. This is because both of these approaches predefine a section of waypoints in the scenario, allowing the NPC vehicle to drive at a constant speed until it reaches the last point, where its speed suddenly drops to zero. Specifically, NSGAII-DT mutates the starting and ending points of the NPC's trajectory as well as its constant speed but fails to improve the interactions effectively. The speeds of the NPC vehicle in AV-Fuzzer is shown in Fig. 8(c). The speeds of NPC vehicles change abruptly several times as the simulation time goes by. This is because the AV-Fuzzer utilizes the speeds of the NPC vehicles as part of the chromosome in the search process, which will produce

Table 2. Efficiency of AdvFuzz Compared with Other Approaches

Metrics	2-lane Urban Way					4-lane Highway
	Rand	NSGAII-DT	AV-Fuzzer	AutoFuzz	AdvFuzz	AdvFuzz
One Scenario (min)	0.80	0.78	1.27	0.79	1.02	1.06
One Violation (min)	7.83	3.41	2.77	4.43	1.33	2.12
One EGO_Fault (min)	144.53	55.38	19.64	26.18	1.53	3.09
First Violation Found (min)	10.32	5.23	20.14	6.72	2.18	3.21
First EGO_Fault Found (min)	74.32	25.53	50.11	18.56	3.23	3.51

a significant speed change in random mutation. Fig. 8(d) shows that the speeds of NPC vehicles in AutoFuzz change from a constant speed to another constant speed suddenly. AutoFuzz also uses waypoints to guide the movement of NPC vehicles, but the speed changes of NPC vehicles are abrupt. In contrast, as shown in Fig. 8(e), the changes in vehicle speed are more diverse and there are few sudden speed changes of adversarial NPC vehicles in AdvFuzz.

Moreover, the predefined speeds of NPC vehicles in Rand, NSGAII-DT, AV-Fuzzer, and AutoFuzz are not related to the speed of the EGO vehicle. That is, the speeds of NPC vehicles will not change according to the speed of the EGO vehicle when interacting, resulting in more aggressive collisions caused by NPC vehicles. In comparison, the speed changes of adversarial NPC vehicles in AdvFuzz are related to those of the EGO vehicle. This is because the adversarial NPC vehicles in AdvFuzz monitor the behaviors of the EGO vehicle and adjust their speeds, thereby increasing the interactions with the EGO vehicle.

Summary. AdvFuzz generates 198.34% more violations than other approaches and increases the proportion of violations caused by the EGO vehicle to 87.04%, which is more than 7 times that of other approaches. Besides, AdvFuzz can find more types of EGO_Fault in 12 hours and the speed changes of NPC vehicles in AdvFuzz are more diverse and reasonable. Therefore, AdvFuzz can effectively improve the interactions between the EGO vehicle and the NPC vehicles and regulate the behavior of NPC vehicles, maximizing the possibility of violations caused by the EGO vehicle.

3.3 Efficiency Evaluation (RQ2)

Table 2 presents the efficiency of AdvFuzz in generating safety violations compared to other approaches. With respect to the time to generate one scenario, NSGAII-DT is the fastest among all approaches, taking 0.78 minutes, while AV-Fuzzer is the slowest, taking 1.27 minutes. AdvFuzz takes 1.02 minutes to generate one scenario in the two-lane urban way, and 1.06 minutes in the four-lane highway. It can be seen that the average time for AdvFuzz to generate one scenario is not much slower than other approaches.

With respect to the time to find one violation, Rand, NSGAII-DT, AV-Fuzzer and AutoFuzz take 7.83 minutes, 3.41 minutes, 2.77 minutes and 4.43 minutes in the 2-lane urban way respectively, while AdvFuzz is at least 51.98% and at most 83.01% faster than other approaches, taking 1.33 minutes. Besides, in the four-lane highway, AdvFuzz takes 2.12 minutes to find one violation.

With respect to the time to find one EGO_Fault, the other approaches take 61.93 minutes in the 2-lane urban way on average, with AV-Fuzzer being the fastest, taking 19.64 minutes, and Rand being the slowest, taking 144.53 minutes. AdvFuzz takes 1.53 minutes on average to find one EGO_Fault in the two-lane urban way, which is at least 92.21% and at most 98.94% faster than other approaches, and 3.09 minutes on average in the four-lane highway.

With respect to the time to find the first violation, Rand, NSGAII-DT, AV-Fuzzer and AutoFuzz take 10.32 minutes, 5.23 minutes, 20.14 minutes and 6.72 minutes in the 2-lane urban way respectively. AdvFuzz finds the first violation with 2.18 minutes, while other approaches take 10.60

Table 3. The Effect of Different Values of ℓ on the Performance of AdvFuzz

Metrics	2-lane Urban Way			4-lane Highway		
	AdvFuzz-20	AdvFuzz-30	AdvFuzz-40	AdvFuzz-20	AdvFuzz-30	AdvFuzz-40
Scenario_Num	708	699	546	681	576	459
Violation_Num	540	402	143	340	272	128
EGO_Fault_Num	470	358	129	233	202	100
Proportion	87.04	89.05	90.21	68.53	74.26	78.13
One Scenario (min)	1.02	1.03	1.32	1.06	1.25	1.57
One Violation (min)	1.33	1.79	5.03	2.12	2.65	5.63
One EGO_Fault (min)	1.53	2.01	5.58	3.09	3.56	7.20
First Violation Found (min)	2.18	3.44	6.51	3.21	4.37	7.12
First EGO_Fault Found (min)	3.23	4.54	6.59	3.51	5.24	8.34

minutes on average. In addition, AdvFuzz takes 3.21 minutes to find the first violation scenario in the 4-lane highway.

With respect to the time to find the first violation caused by the EGO vehicle, AutoFuzz takes 18.56 minutes, which is the fastest among other four approaches and Rand is the slowest, using 74.32 minutes. AdvFuzz uses 3.23 minutes, while other approaches take 41.97 minutes on average. In the four-lane highway, AdvFuzz takes 3.51 minutes to find the first violation caused by the EGO vehicle.

Summary. AdvFuzz takes 1.04 minutes to generate one scenario. AdvFuzz takes 1.73 minutes to find one violation and 2.31 minutes to find one EGO_Fault, which is at least 51.98% and 92.21% faster than those of the other four approaches respectively. In addition, AdvFuzz is at least 58.32% faster in finding the first violation and 82.60% faster in finding the first violation caused by the EGO vehicle than those of the other approaches. Therefore, AdvFuzz is efficient to find violations caused by EGO vehicle in simulation testing.

3.4 Parameter Sensitivity Evaluation (RQ3)

Table 3 reports the effect of different values of ℓ on the performance of AdvFuzz in 2-lane urban way and 4-lane highway scenarios. With respect to the effect on effectiveness of AdvFuzz, we observe that the number of scenarios generated in 12 hours decreases from 708 to 546 in 2-lane urban way and decreases from 681 to 459 in 4-lane highway as the value of ℓ increases from 20 to 40. The number of violations detected by AdvFuzz decreases from 540 to 143 in 2-lane urban way and decreases from 340 to 128 in 4-lane highway. This is because the larger value of ℓ, the greater reaction distance is given to the EGO vehicle. The EGO vehicle is more likely to go through the experimental field without any violation, taking more time to execute a scenario on average. Moreover, the number of EGO_Fault decreases from 470 to 129 in 2-lane urban way and decreases from 233 to 100 in 4-lane highway. The proportion of violation scenarios caused by the EGO vehicle increases as the value of ℓ increases. This is because the larger the value of ℓ, adversarial NPC vehicles are less likely to change lanes suddenly, resulting in less violations caused by NPC vehicles.

With respect to the effect on efficiency of AdvFuzz, we observe that the average time taken to generate one scenario increases from 1.02 minutes to 1.32 minutes in 2-lane urban way and from 1.06 minutes to 1.57 minutes in 4-lane highway as the value of ℓ increases. The average time taken to find one violation increases from 1.33 minutes to 5.03 minutes in 2-lane urban way and from 2.12 minutes to 5.63 minutes in 4-lane highway. The average time taken to find one EGO_Fault increases from 1.53 minutes to 5.58 minutes in 2-lane urban way and from 3.09 minutes to 7.20 minutes in 4-lane highway. As the value of ℓ increases, the EGO vehicle has a longer reaction distance, which allows the EGO vehicle to navigate through the bubble with fewer sudden maneuvers and interactions. This leads to an overall longer simulation duration and increases the time needed to find a violation or EGO_Fault. In addition, with the larger value of ℓ, AdvFuzz seems to need more time to find the first violation and the first EGO_Fault.

Summary. The value of ℓ has a significant impact on the performance of AdvFuzz. A larger value of ℓ results in fewer scenarios generated, fewer violations detected, higher proportion of EGO_Fault. Besides, as the value of ℓ increases, the average time taken to generate one scenario, find one violation and find one EGO_Fault increases as well as the time to find the first violation and the first EGO_Fault.

4 Threats to Validity

First, the selection of target ADS and simulator poses a threat to validity. We select Apollo as our target ADS which is an open-source ADS and widely used in the industry. We choose LGSVL because it has good compatibility with Apollo. However, some modules in Apollo may suffer from high delays due to performance degradation after long-time continuous simulation, which may lead to violations, resulting in false-positive results. We restart the modules of Apollo periodically and use high-performance computers for experiments to mitigate this threat.

Second, the selection of baselines poses another threat to validity. To mitigate this threat, we implement a random approach and select three state-of-the-art approaches that support Apollo and LGSVL. NSGAII-DT uses decision trees to guide the generation. AV-Fuzzer is based on a fuzzing engine using genetic algorithm. AutoFuzz is one of the newest testing approaches guided by neural network. Thus, we believe our selected baselines are representative. We do not compare AdvFuzz with other approaches due to differences in experimental configurations and environments, or because they can not be fully reproduced.

Third, the rule-based Liability Determiner may not be comprehensive enough to diagnose all the collision scenarios correctly. Thus, we select 268 violation scenarios from 880 in total, achieving a confidence level of 95% and a margin error of 5%. We ask two of the authors to check the diagnosis results separately and the Cohen Kappa coefficient reaches 0.845. Finally, the accuracy of the Liability Determiner reaches 91.79%, while both the precision and recall for identifying EGO_Fault are approximately 90.91%, the precision and recall for identifying NPC_Fault are approximately 76.09%. Thus, we believe the results given by Liability Determiner are convincing.

Last, the subjective diagnosis and classification of violations caused by EGO vehicle affects the validity. To mitigate this threat, we ask another two of the authors to classify the violation scenarios caused by EGO vehicle into different types in terms of vehicles' behaviors and the reasons for violations. They separately diagnose and classify each violation scenario caused by EGO vehicle. If the two authors' decisions conflict, a third author is involved for a group discussion to reach agreements. Finally, the Cohen Kappa coefficient reaches 0.862.

5 Related Work

Scenario Description Language. Multiple tool-independent DSLs have been proposed for testing ADSs, providing a formal definition of scenario structure and vehicle behavior. For example, Scenic [16] characterizes driving scenarios based on a probabilistic programming approach. GeoScenario [48] is a DSL for scenario description to substantiate testing scenarios. Besides, OpenScenario [3] is an XML-based standard, describing dynamic content in driving simulation applications in combination with OpenDRIVE [4]. Recently, Queiroz et al. [49] present SDV model to express and execute scenarios for ADS testing in simulation, providing a user-oriented language to coordinate the vehicle behavior and motion planning that optimizes for realism and achieving the scenario test objective. The varying structure and syntax of these DSLs require significant time to master, whereas AdvFuzz is user-friendly and ready to use out of the box.

Scenario-Based Testing. Scenario-based testing [14, 22, 37, 38, 58, 61] has been widely studied to generate diverse driving scenarios for ADS testing to identify safety violations. Numerous works [1, 8, 36, 53, 54] use a genetic algorithm-based approach to generate scenarios where the EGO vehicle may collide with NPC vehicles while a few works [2, 12, 20, 26, 31, 41] guide the ADSs to violate predefined rules, such as failing to reach their destination, or to exhibit incorrect behaviors like speeding or executing unsafe lane changes. Sun et al. [51], Zhang et al. [59] and Li et al. [35] propose to generate driving scenarios that break specific traffic rules. Huai et al. [25] focus on generating valid and effective driving scenarios that lead to comfort and safety violations. Additionally, Lu et al. [39], Zhong et al. [60] and Wang et al. [55] employ neural network or reinforcement learning to guide the generation of scenarios. Besides, a few works [7, 13, 17, 57] attempt to reproduce real-world data (e.g., traffic accident reports and vehicle trajectories) to find corner cases in simulation. Several works [9, 45, 56] investigate the metrics (e.g., physical environment-state coverage metric [23]) in simulation to guide the generation. Lu et al. [40] and Chen et al. [11] study the configuration of simulation in ADS testing.

To the best of our knowledge, all the scenarios generated by these previous approaches are static and lack adaptability. Consequently, they are usually inefficient in generating challenging scenarios for ADS testing and fail to reduce the number of violation scenarios caused by NPC vehicles. Our work aims to generate more interactive adversarial scenarios, where NPC vehicles can make maneuver decisions according to ADSs' behaviors, leading to more violation scenarios caused by ADS. Huai et al. [26] try to maximize the violations caused by ADS, they opt to bridge multiple ADSs for interaction rather than using NPC vehicles. However, this is achieved at the cost of feeding ground truth directly into the ADSs' localization and perception modules and only testing the planning module. The idea closest to our work is that of Chen et al. [10], who design an adaptive evaluation framework to find crashes in adversarial environments generated by deep reinforcement learning. However, they only focus on lane-change scenarios and fail to connect the ADS with simulator. Differently, we propose a new framework to generate adversarial scenarios and test ADSs at the system-level.

Behavior Tree in Simulation Testing. Behavior tree is a modular, scalable, discrete control architecture, overcoming the limitations of finite state machines and their variants [18, 19, 27]. Several works have suggested using behavior trees in simulation testing. For example, BTScenario [29] employs behavior trees to provide driving control inputs directly to longitudinal and lateral controllers. However, it lacks a trajectory planner, making it impossible to plan flexible and realistic trajectories. Larter et al. [32] utilize behavior trees to control pedestrians (i.e., setting motion objectives) in simulation. As far as we know, no work combines behavior trees with the maneuver decisions of NPC vehicles to create an adversarial scenario in ADS simulation testing.

6 Conclusion

In order to enhance the interaction between the EGO vehicle and NPC vehicles and regulate the behaviors of NPC vehicles, we have proposed adversarial NPC vehicles and implemented AdvFuzz to automatically generate adversarial scenarios on main lanes (e.g., urban ways and highways) for ADS simulation testing, maximizing the possibility of violation scenarios caused by the EGO vehicle. Large-scale experiments have been conducted to demonstrate the effectiveness and efficiency of AdvFuzz. In the future, we plan to extend AdvFuzz to support more ADSs and simulators. Moreover, we also plan to support more types of roads such as intersections and roundabouts.

7 Data Availability

All the experimental data and source code of our work are available at our replication site https://advfuzz.github.io/AdvFuzz/.

References

[1] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing Vision-Based Control Systems Using Learnable Evolutionary Algorithms. In *Proceedings of the IEEE/ACM 40th International Conference on Software Engineering*. 1016–1026.

[2] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C Briand, and Thomas Stifter. 2018. Testing autonomous cars for feature interaction failures using many-objective search. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 143–154.

[3] ASAM. 2021. *ASAM OpenSCENARIO: User Guide.* Retrieved August 11, 2024 from https://www.asam.net/index.php?eID=dumpFile&t=f&f=4092&token=d3b6a55e911b22179e3c0895fe2caae8f5492467

[4] ASAM. 2022. *ASAM OpenDRIVE.* Retrieved August 11, 2024 from https://www.asam.net/standards/detail/opendrive/

[5] Daniel Atherton. 2022. Incident 434: Sudden Braking by Tesla Allegedly on Self-Driving Mode Caused Multi-Car Pileup in Tunnel. In *AI Incident Database*, Khoa Lam (Ed.). Responsible AI Collaborative. Retrieved February 13, 2023 from https://incidentdatabase.ai/cite/434/

[6] Baidu. 2022. *Apollo: An open autonomous driving platform.* Retrieved August 12, 2024 from https://github.com/ApolloAuto/apollo

[7] Sai Krishna Bashetty, Heni Ben Amor, and Georgios Fainekos. 2020. Deepcrashtest: Turning dashcam videos into virtual crash tests for automated driving systems. In *Proceedings of the 2020 IEEE International Conference on Robotics and Automation*. 11353–11360.

[8] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2016. Testing Advanced Driver Assistance Systems Using Multi-Objective Search and Neural Networks. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. 63–74.

[9] Christian Birchler, Tanzil Kombarabettu Mohammed, Pooja Rani, Teodora Nechita, Timo Kehrer, and Sebastiano Panichella. 2024. How does Simulation-based Testing for Self-driving Cars match Human Perception? *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 929–950.

[10] Baiming Chen, Xiang Chen, Qiong Wu, and Liang Li. 2021. Adversarial evaluation of autonomous vehicles in lane-change scenarios. *IEEE Transactions on Intelligent Transportation Systems* 23, 8 (2021), 10333–10342.

[11] Yuntianyi Chen, Yuqi Huai, Shilong Li, Changnam Hong, and Joshua Garcia. 2024. Misconfiguration Software Testing for Failure Emergence in Autonomous Driving Systems. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 1913–1936.

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

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

[14] Wenhao Ding, Chejian Xu, Mansur Arief, Haohong Lin, Bo Li, and Ding Zhao. 2023. A Survey on Safety-Critical Driving Scenario Generation—A Methodological Perspective. *IEEE Transactions on Intelligent Transportation Systems* 24, 7 (2023), 6971–6988.

[15] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*. 1–16.

[16] Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2019. Scenic: A Language for Scenario Specification and Scene Generation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 63–78.

[17] Alessio Gambi, Tri Huynh, and Gordon Fraser. 2019. Generating effective test cases for self-driving cars from police reports. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 257–267.

[18] Razan Ghzouli, Thorsten Berger, Einar Broch Johnsen, Swaib Dragule, and Andrzej Wąsowski. 2020. Behavior trees in action: a study of robotics applications. In *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering*. 196–209.

[19] Razan Ghzouli, Thorsten Berger, Einar Broch Johnsen, Andrzej Wasowski, and Swaib Dragule. 2023. Behavior trees and state machines in robotics applications. *IEEE Transactions on Software Engineering* 49, 9 (2023), 4243–4267.

[20] Christoph Gladisch, Thomas Heinz, Christian Heinzemann, Jens Oehlerking, Anne von Vietinghoff, and Tim Pfitzer. 2020. Experience Paper: Search-Based Testing in Automated Driving Control Applications. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*. 26–37.

[21] Fitash Ul Haq, Donghwan Shin, and Lionel Briand. 2022. Efficient online testing for DNN-enabled systems using surrogate-assisted and many-objective optimization. In *Proceedings of the 44th international conference on software engineering*. 811–822.

[22] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel C. Briand. 2020. Comparing Offline and Online Testing of Deep Neural Networks: An Autonomous Car Case Study. In *Proceedings of the IEEE 13th International Conference on Software Testing, Validation and Verification*. 85–95.

[23] Carl Hildebrandt, Meriel von Stein, and Sebastian Elbaum. 2023. PhysCov: physical test coverage for autonomous vehicles. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 449–461.

[24] Yuqi Huai. 2023. *SORA-SVL*. Retrieved August 12, 2024 from https://www.ics.uci.edu/~yhuai/SORA-SVL/

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

[26] Yuqi Huai, Yuntianyi Chen, Sumaya Almanee, Tuan Ngo, Xiang Liao, Ziwen Wan, Qi Alfred Chen, and Joshua Garcia. 2023. Doppelgänger Test Generation for Revealing Bugs in Autonomous Driving Software. In *Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering*. 2591–2603.

[27] Mais Jamal and Aleksandr Panov. 2021. Adaptive maneuver planning for autonomous vehicles using behavior tree on apollo platform. In *Proceedings of the 41st SGAI International Conference on Artificial Intelligence*. 327–340.

[28] Nidhi Kalra and Susan M Paddock. 2016. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice* 94 (2016), 182–193.

[29] Shuting Kang, Haoyu Hao, Qian Dong, Lingzhong Meng, Yunzhi Xue, and Yanjun Wu. 2022. Behavior-tree based scenario specification and test case generation for autonomous driving simulation. In *Proceedings of the 2nd International Conference on Intelligent Technology and Embedded Systems*. 125–131.

[30] Srishti Khemka. 2021. Incident 347: Waymo Self-Driving Taxi Behaved Unexpectedly, Driving away from Support Crew. In *AI Incident Database*, Khoa Lam (Ed.). Responsible AI Collaborative. Retrieved February 13, 2023 from https://incidentdatabase.ai/cite/347/

[31] Seulbae Kim, Major Liu, Junghwan" John" Rhee, Yuseok Jeon, Yonghwi Kwon, and Chung Hwan Kim. 2022. Drivefuzz: Discovering autonomous driving bugs through driving quality-guided fuzzing. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 1753–1767.

[32] Scott Larter, Rodrigo Queiroz, Sean Sedwards, Atrisha Sarkar, and Krzysztof Czarnecki. 2022. A hierarchical pedestrian behavior model to generate realistic human behavior in traffic simulation. In *Proceedings of the IEEE Intelligent Vehicles Symposium*. 533–541.

[33] LGSVL. 2021. *lgsvl - A Python API for SVL Simulator*. Retrieved August 12, 2024 from https://github.com/lgsvl/PythonAPI

[34] LGSVl. 2021. *SVL Simulator: An Autonomous Vehicle Simulator*. Retrieved August 12, 2024 from https://github.com/lgsvl/simulator/releases/tag/2021.3

[35] Changwen Li, Joseph Sifakis, Qiang Wang, Rongjie Yan, and Jian Zhang. 2023. Simulation-Based Validation for Autonomous Driving Systems. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 842–853.

[36] Guanpeng Li, Yiran Li, Saurabh Jha, Timothy Tsai, Michael Sullivan, Siva Kumar Sastry Hari, Zbigniew Kalbarczyk, and Ravishankar Iyer. 2020. Av-fuzzer: Finding safety violations in autonomous driving systems. In *Proceedings of the 2020 IEEE 31st International Symposium on Software Reliability Engineering*. 25–36.

[37] Li Li, Wu-Ling Huang, Yuehu Liu, Nan-Ning Zheng, and Fei-Yue Wang. 2016. Intelligence testing for autonomous vehicles: A new approach. *IEEE Transactions on Intelligent Vehicles* 1, 2 (2016), 158–166.

[38] Guannan Lou, Yao Deng, Xi Zheng, Mengshi Zhang, and Tianyi Zhang. 2022. Testing of autonomous driving systems: where are we and where should we go?. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 31–43.

[39] Chengjie Lu. 2023. Test Scenario Generation for Autonomous Driving Systems with Reinforcement Learning. In *Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings*. 317–319.

[40] Chengjie Lu, Yize Shi, Huihui Zhang, Man Zhang, Tiexin Wang, Tao Yue, and Shaukat Ali. 2022. Learning configurations of operating environment of autonomous vehicles to maximize their collisions. *IEEE Transactions on Software Engineering* 49, 1 (2022), 384–402.

[41] Yixing Luo, Xiao-Yi Zhang, Paolo Arcaini, Zhi Jin, Haiyan Zhao, Fuyuki Ishikawa, Rongxin Wu, and Tao Xie. 2021. Targeting requirements violations of autonomous driving systems by dynamic evolutionary search. In *Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering*. 279–291.

[42] Interactive Mathematics. 2024. *Velocity (s-t) Graphs*. Retrieved August 12, 2024 from https://www.intmath.com/kinematics/1-velocity-graphs.php

[43] Sean McGregor. 2018. Incident 4: Uber AV Killed Pedestrian in Arizona. In *AI Incident Database*, Sean McGregor (Ed.). Responsible AI Collaborative. Retrieved February 13, 2023 from https://incidentdatabase.ai/cite/4

[44] Michael E Mortenson. 1999. *Mathematics for computer graphics applications*. Industrial Press Inc.

[45] Neelofar Neelofar and Aldeida Aleti. 2024. Towards reliable ai: Adequacy metrics for ensuring the quality of system-level testing of autonomous vehicles. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–12.

[46] National Committee on Uniform Traffic Laws. 1952. *Uniform vehicle code*. Vol. 5. Department of Commerce, Bureau of Public Roads.

[47] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. 2016. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles* 1, 1 (2016), 33–55.

[48] Rodrigo Queiroz, Thorsten Berger, and Krzysztof Czarnecki. 2019. GeoScenario: An Open DSL for Autonomous Driving Scenario Representation. In *Proceedings of the IEEE Intelligent Vehicles Symposium*. 287–294.

[49] Rodrigo Queiroz, Divit Sharma, Ricardo Caldas, Krzysztof Czarnecki, Sergio García, Thorsten Berger, and Patrizio Pelliccione. 2024. A driver-vehicle model for ADS scenario-based testing. *IEEE Transactions on Intelligent Transportation Systems* 25, 8 (2024), 8641–8654.

[50] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, et al. 2020. LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving. *arXiv preprint arXiv:2005.03778* (2020).

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

[52] Tadewos G Tadewos, Laya Shamgah, and Ali Karimoddini. 2019. Automatic safe behaviour tree synthesis for autonomous agents. In *Proceedings of the IEEE 58th Conference on Decision and Control*. 2776–2781.

[53] Haoxiang Tian, Yan Jiang, Guoquan Wu, Jiren Yan, Jun Wei, Wei Chen, Shuo Li, and Dan Ye. 2022. MOSAT: finding safety violations of autonomous driving systems using multi-objective genetic algorithm. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 94–106.

[54] Haoxiang Tian, Guoquan Wu, Jiren Yan, Yan Jiang, Jun Wei, Wei Chen, Shuo Li, and Dan Ye. 2022. Generating Critical Test Scenarios for Autonomous Driving Systems via Influential Behavior Patterns. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.

[55] Tong Wang, Taotao Gu, Huan Deng, Hu Li, Xiaohui Kuang, and Gang Zhao. 2024. Dance of the ADS: Orchestrating Failures through Historically-Informed Scenario Fuzzing. *arXiv preprint arXiv:2407.04359* (2024).

[56] Trey Woodlief, Felipe Toledo, Sebastian Elbaum, and Matthew B Dwyer. 2024. S3C: Spatial Semantic Scene Coverage for Autonomous Vehicles. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[57] Xudong Zhang and Yan Cai. 2023. Building critical testing scenarios for autonomous driving from real accidents. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 462–474.

[58] Xinhai Zhang, Jianbo Tao, Kaige Tan, Martin Törngren, José Manuel Gaspar Sánchez, Muhammad Rusyadi Ramli, Xin Tao, Magnus Gyllenhammar, Franz Wotawa, Naveen Mohan, Mihai Nica, and Hermann Felbinger. 2023. Finding Critical Scenarios for Automated Driving Systems: A Systematic Mapping Study. *IEEE Transactions on Software Engineering* 49, 3 (2023), 991–1026.

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

[60] Ziyuan Zhong, Gail Kaiser, and Baishakhi Ray. 2023. Neural network guided evolutionary fuzzing for finding traffic violations of autonomous vehicles. *IEEE Transactions on Software Engineering* 49, 4 (2023), 1860–1875.

[61] Ziyuan Zhong, Yun Tang, Yuan Zhou, Vania de Oliveira Neves, Yang Liu, and Baishakhi Ray. 2021. A survey on scenario-based testing for automated driving systems in high-fidelity simulation. *arXiv preprint arXiv:2112.00964* (2021).

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

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009