USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

Short: APSFuzz: Simulation-Based Fuzzing Testing for Automated Parking Systems

Tong Bu, Jiarun Dai, Jiaqi Luo, Songyang Peng, Zongan Huang, and Min Yang, *Fudan University*

https://www.usenix.org/conference/vehiclesec25/presentation/bu

This paper is included in the Proceedings of the 3rd USENIX Symposium on Vehicle Security and Privacy.

August 11–12, 2025 • Seattle, WA, USA

978-1-939133-49-6

Open access to the Proceedings of the 3rd USENIX Symposium on Vehicle Security and Privacy is sponsored by USENIX.

APSFuzz: Simulation-Based Fuzzing Testing for Automated Parking Systems

Tong Bu*
Fudan University, China

Jiarun Dai*
Fudan University, China

Jiaqi Luo
Fudan University, China

Songyang Peng
Fudan University, China

Zongan Huang
Fudan University, China

Min Yang
Fudan University, China

Abstract

Automated Parking System (APS) is a modern vehicle-equipped AI system that automates the process of parking vehicles. Nowadays, various companies (e.g., Tesla) have already deployed APSs on their latest released vehicles. Given the popularity of APSs, however, real-world APS misbehaviors (e.g., collision) continue to occur, calling for reliable techniques for the robustness testing of APSs. Existing works generally focus on safety testing of Autonomous Driving Systems (ADS) on public roads, which cannot comply with the unique characteristics of parking scenarios (e.g., vehicle behaviors and testing criteria). In light of this, we propose APSFuzz, a novel simulation-based APS fuzzer to effectively detect bugs that result in misbehaviors (e.g., collision, stuck, pose error, etc.) of APS. Based on the systematic modeling of parking scenarios, APSFuzz leverages parking-scenario-specific mutation strategies and a scheduling mechanism to ensure the effectiveness of fuzzing-based simulation testing. In the evaluation, we built the prototype of APSFuzz based on the Carla simulator to identify the robustness flaws of Autoware.Universe (i.e., an open-source APS). Finally, APSFuzz helped identify 74 buggy parking scenarios for Autoware.Universe, caused by 5 types of root causes. We have reported these 5 root causes to the developers, and till now 1 of them has been patched.

1 Introduction

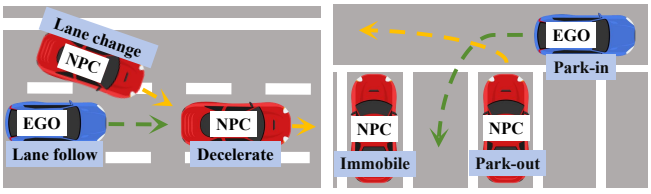
Automated Parking System (APS) is an AI-based driver assistance system that automates the process of parking vehicles, from locating available parking spaces to entering and exiting the garage or lot. In recent years, the global market for APS has witnessed remarkable growth and APS has undoubtedly become a standard feature in modern vehicles [1]. However, accidents [2] caused by APSs (e.g., collisions and scratches) continue to occur, significantly hindering the widespread commercial deployment of APSs. In this state of confusion, manufacturers are in severe need of reliable techniques to identify safety or functionality flaws of APS.

To date, various testing methodologies have been proposed to assess the robustness of Autonomous Driving Systems (ADS). Particularly, fuzzing-based simulation testing [9, 15, 17–20, 23, 24, 26, 27] has been embraced as a fundamental technique to identify safety or functionality flaws of ADSs. Generally, this line of technique works by combining fuzzing testing with simulation testing, automatically mutating configurable parameters of simulation scenarios, and thus mining driving conditions that are prone to trigger ADS flaws. Simulation-based fuzzing has demonstrated high effectiveness in identifying ADS misbehaviors (e.g., collision and traffic law violations) on public roads (e.g., urban streets, highways, and intersections). However, to the best of our knowledge, there are no existing works designed for APS testing. In light of this, it should be an appealing solution to migrate public-road-specific testing methodologies to the parking scenario, so as to facilitate the APS robustness assessment.

However, there exist two main differences between parking scenarios and public road scenarios, which hinder the direct migration of existing techniques [9,15,17–20,23,24,26,27] to APS testing. ❶ *Different Vehicle Behaviors*. For autonomous vehicles, parking scenarios have greater freedom of movement without roadline restrictions, indicating an even more challenging scenario search space for APS robustness testing. To be more specific, parking scenarios involve vehicle behaviors (e.g., parking in and out) that are not systematically modeled in previous works. Hence, existing practices provide limited guidance about how to traverse the parking scenario search space. ❷ *Different Driving Tasks and Testing Criterias*. Previous studies on ADS testing primarily consider safety-critical testing objectives (e.g., collision and traffic law violation). However, to faithfully investigate the robustness of APSs, not only the safety-critical criteria but also parking-specific functional criteria (e.g., the vehicle's posture after parking is completed) should be considered. This critical distinction renders existing fuzzing scheduling schemes inadequate for direct application to parking scenarios.

*Co-first authors.

Challenges. As mentioned above, considering the unique characteristics of parking scenarios, we are motivated to design an APS-specific fuzzing pipeline, so as to identify potential robustness flaws of target APS. To achieve this goal, the following two challenges should be addressed: ❶ *How to curate critical vehicle interactions between NPCs (i.e., simulated surrounding vehicles around the autonomous vehicle) and EGO (i.e., the autonomous vehicle controlled by the APS under test) in parking scenario?* Common practices suggest that critical vehicle interactions (e.g., surrounding vehicle is aggressively overtaking) are essential causes of ADS-involved accidents [18]. As illustrated in Fig. 1(A), due to lane restrictions, EGO usually has to drive along a specific lane, which allows NPC vehicles in public road scenarios to easily interact with EGO by performing maneuvers such as lane changes and deceleration. But parking scenarios are kind of free spaces, which means there are no lane markings to constrain its movement. Theoretically, the trajectories planned by EGO can take totally different forms (see Fig. 1(B)) compared to those on public roads. Hence, it is hard to generate high-quality simulation scenarios in which NPC vehicles critically interact with EGO. ❷ *How to efficiently identify diverse robustness flaws of target APS?* Generally, APS testing has unique and diverse testing criteria (e.g., parking pose error), which have not been considered among public-road-specific testing methodologies [17–19, 24]. In light of this, we should design APS-specific testing oracles and fuzzing scheduling mechanisms.



(A) vehicle interactions on public roads (B) vehicle interactions in parking lots

Figure 1: Different Vehicle Interaction Patterns in Different Driving Scenarios.

Our Work. To address the above challenges, we propose APSFuzz, a novel simulation-based APS fuzzer. First, APSFuzz integrates an offline-online hybrid methodology to generate critical NPC-EGO vehicle interactions in parking scenarios. Our key observation hints that EGO is likely to plan a similar trajectory when the scenario conditions remain unchanged. Therefore, before each trial of parking scenario mutation (i.e., the offline phase), we collect EGO trajectory from the previous execution of this scenario as a reference and generate NPCs whose trajectories have interactions with the EGO trajectory. After that, during runtime execution of the mutated scenario, we further timely adjust the speed of NPCs to ensure critical EGO-NPC interactions. Besides, APSFuzz also systematically implements four testing oracles for APS fuzzing (e.g., collision, stuck, timeout, and pose error), combined with a fuzzing scheduling mechanism that can balance the efficiency and diversity of APS flaw identification.

Contributions. In summary, this paper makes the following contributions:

- We propose an interaction-oriented parking scenario mutation method, which enhances the interaction between NPCs and EGO by generating static files offline and adjusting NPC speeds at runtime, thereby increasing the critical level of the scenario.
- We propose a multi-objective seed scheduling scheme that allows the efficient exploration of diverse buggy parking scenarios.
- We implement the prototype of APSFuzz and identify 5 unique root causes of Autoware, each of which would cause APS misbehaviors. We open-source the source code of APSFuzz (https://github.com/JSGforever/APSFuzz), as well as the testing environment and data, to ease follow-up research.

2 Approach

2.1 Framework Overview

As shown in Figure 2, APSFuzz comprises three essential components, namely Interaction-Oriented Scenario Mutation (see §2.3), Misbehavior Detector (see §2.4) and Multi-objective Seed Scheduling (see §2.5).

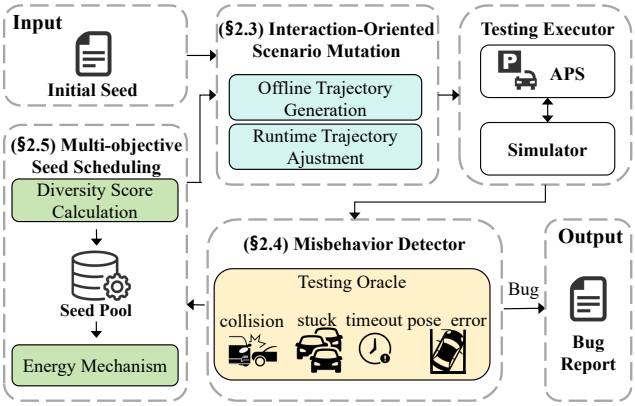


Figure 2: Workflow of APSFuzz

2.2 Parking Scenario Formalization

Symbol Definition. We first define necessary symbols to ease the clarification of our methodology. In order to describe the possible NPC behaviors in parking scenarios, we define some configurable parameters including $\mathbb{IP}$ ("init position"),

Table 1: NPC behavior modeling.

NPC Type	Behavior Type	Configurable Parameters					
		$\mathbb{IP}$	$\mathbb{GP}$	$\mathbb{MS}$	$\mathbb{TP}$	$\mathbb{D}$	$\mathbb{PP}$
Vehicle	park-in	✓	✓	✓	✓		✓
	park-out	✓	✓	✓	✓		✓
	immobile	✓					
Pedestrain	linear	✓	✓	✓		✓	
	immobile	✓					
Barrier	immobile	✓					

Definitions of $\mathbb{IP}$, $\mathbb{GP}$, $\mathbb{MS}$, $\mathbb{TP}$, $\mathbb{D}$ and $\mathbb{PP}$ are clarified at the beginning of §2.2.

initial position of NPC; $\mathbb{GP}$ ("goal position"), goal position for dynamic NPC; $\mathbb{MS}$ ("max speed"), maximum speed of NPC; $\mathbb{TP}$ ("turning point"), position of the NPC vehicle to switch on or off the parking mode (i.e., follow the trajectory planned by freespace planning algorithm, detailed in §2.3); $\mathbb{D}$ ("delay"), the delay before the dynamic pedestrian movement; $\mathbb{PP}$ ("parking pattern"), whether NPC vehicle reverses into the parking slot or enters it head-on. Lastly, $\mathbb{NT}$ and $\mathbb{BT}$ refer to NPC type and behavior type of NPCs.

NPC Behaviors Modeling. We categorize an NPC's behavior in a parking scenario into three parts (shown in Table 1): ❶ *NPC Type* simply refers to the categories of NPCs (e.g., vehicles, pedestrians). ❷ *Behavior Type* defines behaviors for specific NPC types (e.g., parking in for NPC vehicles). Moreover, we propose a set of ❸ *Configurable Parameters* to execute a complete and deterministic behavior. The introduction of configurable parameters provides a large input space for scenario mutations. By combining the three components, all NPC behaviors can be formally represented as a vector as follows:

$$M = \{\mathbb{NT}, \mathbb{BT}, \{\mathbb{IP}, \mathbb{GP}, \mathbb{MS}, \mathbb{TP}, \mathbb{D}, \mathbb{PP}\}\}$$

Each vector can represent a unique NPC, and these parameters can be used for scenario mutation.

2.3 Interaction-Oriented Scenario Mutation

In this phase, we aim to continuously modify the scenarios to enhance interactions with EGO (e.g., increasing the interaction between NPC trajectories and EGO trajectory). However, as discussed in §1, due to the lack of lane constraints, predicting EGO trajectory in a parking scenario solely based on the initial and target points remains highly challenging. Fortunately, we observed that EGO is likely to plan a similar trajectory when the scenario conditions remain unchanged. Therefore, we can infer EGO trajectory based on its trajectory in the previous scenario. Leveraging this observation, we implemented an offline-runtime combination strategy to perform scenario mutations.

Offline NPC Trajectory Generation. This strategy ensures that the generated NPC trajectories intersect with EGO trajectory from the previous execution of the scenario. To achieve this goal, we designed a configurable parameter setting method tailored for different $\mathbb{NT}$. Firstly, for NPC vehicles, we randomly select their configurable parameters (shown in Table 1). After that, we will generate trajectories based on the $\mathbb{BT}$ of NPCs. Specifically, if $\mathbb{BT}$ is 'park-in/out', their trajectory consists of two parts (from $\mathbb{IP}$ to $\mathbb{TP}$ and from $\mathbb{TP}$ to $\mathbb{GP}$). Take park-in type vehicles as an example, we first plan a cruising trajectory based on $\mathbb{IP}$, $\mathbb{TP}$, and the information of the map. Then Hybrid A* algorithm (a commonly used parking trajectory planning algorithm [25]) is used to plan the parking trajectory based on $\mathbb{TP}$ and $\mathbb{GP}$. If the $\mathbb{BT}$ is 'immobile', the trajectory is set to the vehicle's initial position. Once the trajectory is generated, we determine whether the NPC trajectory interacts with EGO trajectory. If not, repeat the above process until an interaction is found or the maximum number of attempts (i.e., 10) is reached. Secondly, for NPC pedestrians and barriers, we can start by randomly selecting a point from EGO trajectory of the last execution as intersection point (denoted as IPT) and use this point to set the NPC parameters. For pedestrians with 'linear' $\mathbb{BT}$, we only need to randomly generate a line segment passing through this point, with the start and end points of the line segment set to the $\mathbb{IP}$ and $\mathbb{GP}$ ($\mathbb{MS}$ and $\mathbb{D}$ are set randomly). For immobile-type pedestrians and barriers, we simply need to add a specific offset to the IPT to generate the corresponding $\mathbb{IP}$.

Runtime NPC Trajectory Adjustment. Although the offline strategy ensures that the NPC and EGO trajectories intersect, this alone is insufficient. In order to ensure that the NPC and EGO reach the IPT within a similar timeframe, we also need to control the NPC's speed. To achieve this, we dynamically adjust the NPC's longitudinal speed and obstacle avoidance behavior by real-time acquisition of EGO information and obstacle data. Before running, APSFuzz will proportionally divide the trajectories of both EGO and NPC from the starting point to the IPT. It then monitors the position of EGO in real-time and adjusts the NPC's speed based on the current position of EGO. If an obstacle exists within a certain range along the NPC's path, the NPC will stop and wait until the conditions allow it to proceed.

2.4 Misbehavior Detector

To comprehensively test the APS, we refer to official APS standards [11, 12] and consider both safety-oriented (i.e., collision) and functionality-oriented misbehaviors (i.e., stuck, timeout, pose error). Moreover, following existing practices in ADS testing [6, 27], we leverage heuristic-based misbehavior identification to categorize similar buggy scenarios (detailed as follows).

- **Collision.** Collisions are the most severe safety incidents in parking scenarios. We record all collisions happened and classify them based on the states of EGO and the NPC

collided with EGO, including ❶ The driving direction of EGO (i.e., forward or backward); ❷ $\mathbb{NT}$ of the NPC that collides with EGO; ❸ The state of the NPC that collides with EGO (i.e., immobile, approaching or receding).

- **Stuck.** If the APS does not make any driving decisions for an extended period (i.e., 50 *sec* in this paper) during the parking process, we think the APS has encountered a stuck issue. We classify the stuck scenarios according to NPCs near EGO (i.e., less than 0.5m) at the stuck point. To be more specific, the classification metrics: ❶ The number and $\mathbb{NT}$ of NPCs nearby; ❷ The relative positions between NPCs and EGO (i.e., left, right, front, behind, left-front, right-front, left-behind or right-behind)

- **Timeout.** If APS cannot finish the whole parking task within a given time limit (i.e., 120 *sec* in this paper), we think the target APS has encountered a timeout issue due to insufficient functionalities (i.e., inefficient parking solutions in complex scenarios). Since a timeout scenario may be caused by the combined actions of multiple NPCs in the parking scenario, we record information for all NPCs for timeout scenario classification, including the number, $\mathbb{NT}$ and $\mathbb{BT}$ of NPCs.

- **Pose error.** The final orientation of the vehicle after parking is also very important. If the angle between the front of the vehicle and the side boundary line of the parking space is too large upon completion of parking (with a threshold set at 15 degrees in this paper), it is also considered a functionality issue of the APS. Here, we classify this kind of scenario based on EGO's $\mathbb{PP}$ (i.e., head-on or reverse) and the specific angle between EGO and the parking line (i.e., -90°- -45°, -45°- -15°, 15°- 45° or 45°- 90);

2.5 Multi-objective Seed Scheduling

In this component, we follow BehAVExplor [9] to evaluate the diversity of EGO behavior to maintain an "interesting" seed corpus and then select seeds using an energy-based mechanism. However, it requires collecting a large number of scenarios to train a model that represents the behavior of the EGO, which is time-consuming. Moreover, it doesn't take into account the diversity of buggy scenarios when calculating the energy of the seeds. In our single fuzzing loop, the starting and ending points of the EGO are fixed, meaning that any difference in trajectory (e.g., coordinates) between two scenarios indicates a variation in EGO behavior. Based on this, we redesign the diversity calculation method and integrate the diversity of buggy scenarios into the energy mechanism.

Diversity Score Calculation. Let the recorded EGO trajectories be denoted as $T^{(n)}$, where n is the lengths of a trajectory, T contains coordinate information as well as timestamps allowing each trajectory to represent a complete driving state of EGO. We then use an extended Dynamic Time Warping (DTW) algorithm [7] to calculate the distance between two EGO trajectories, serving as diversity score, given by:

$$d(T_1, T_2) = \min_{\pi} \sum_{i=1}^{n} \text{dist}(T_1^{(i)}, T_2^{(\pi(i))})$$

where π is the warping path, and $\text{dist}(T_1^{(i)}, T_2^{(j)})$ represents the distance function (i.e., the Euclidean distance) between points in T_1 and T_2. A new seed is retained if its diversity score from all scenarios in the seed queue exceeds a preset threshold (i.e., 100). Additionally, we perform a coarse-grained comparison of NPCs by examining their $\mathbb{NT}$ and $\mathbb{BT}$. Specifically, if $\mathbb{NT}$ of more than one NPC or $\mathbb{BT}$ of more than two NPCs are different between the new scenario and the existing one, we will also add the new scenario to the seed queue.

Energy Mechanism. APSFuzz updates the energy of a seed every time when a new seed is added to the seed queue. The energy is measured from the following two aspects: ❶ *Average Scenario Diversity Score.* Let the seed queue be denoted as $S = \{s_1, s_2, \ldots, s_k\}$, where each s_i represents a seed. The average difference between the current seed and all other seeds in the queue can be calculated as :

$$D_{\text{avg}}(s_i) = \frac{1}{k-1} \sum_{\substack{j=1 \\ j \neq i}}^{k} d(s_i, s_j)$$

where: k is the number of seeds in the seed queue, d is the diverse calculation function. ❷ *Selection Frequency Score.* To mitigate local convergence, we introduce the Selection Frequency Score (SF). All newly generated seeds are assigned the same initial SF (i.e., 5). Each time a seed is selected for mutation, its SF is adjusted based on the execution results. Let s_i' represent the new seeds mutated from s_i and $\mathcal{E}_{new}$ is a new misbehavior discovered. If s_i' leads to the discovery of a new misbehavior, the SF of s_i remains unchanged; otherwise, it is reduced. The adjustment follows the formula:

$$SF(s_i) = \begin{cases} SF(s_i) & \text{result}(s_i') = \mathcal{E}_{new}, \\ SF(s_i) - 1 & \text{otherwise} \end{cases}$$

Finally, we use a roulette selection based on the energy of each seed, which is updated as:

$$E(s_i) = SF(s_i) \times D_{\text{avg}}(s_i)$$

2.6 Fuzzing Pipeline

Algorithm 1 shows the fuzzing pipeline of APSFuzz. For each seed scenario, a fuzzing pipeline will be executed (line 1-4). During the fuzzing process, APSFuzz maintains a seed queue. In the initialization phase, since the number of elements in the seed queue has not reached the set threshold (i.e., twice N_p), we mutate the initial seeds to generate new scenarios (line 7-8). Once the threshold is reached, we use

Algorithm 1 Fuzzing Pipeline

Input: S - Initial Seed Pool, N_s - Maximum Scenario Num, N_p - Size of Population
Output: s' - Buggy Scenario Pool

```
1:  for each seed in S do
2:      fuzz_loop(seed)
3:  procedure FUZZ_LOOP(seed)
4:      tested_scenarios ← []
5:      seed_queue ← []
6:      while tested_scenarios.size < N_s do
7:          if seed_queue.size() < 2 * N_p then
8:              selected_seeds ← seed                    ▷ initialization phase
9:          else
10:             selected_seeds ← seed_selection(seed_queue, N_p)
11:         p_list ← scenario_mutate(selected_seeds)
12:         for each scenario in p_list do
13:             simulate(scenario)          ▷ record ego trajectory during test
14:             if scenario.error == True then
15:                 add scenario to s'
16:             if calculate_diversity(scenario, seed_queue) > 100 then
17:                 add scenario to seed_queue
18:             update(seed_queue)                   ▷ update score & energy
19:             add scenario to tested_scenarios
```

a seed selection algorithm to choose a seed from the queue and perform mutations based on the selected seed (line 10). The newly mutated scenarios are sequentially executed in the simulator, and EGO trajectory is recorded (line 13). The misbehavior detector continuously monitors the simulation process to determine whether the scenario is buggy. (line 14-15). After each scenario execution, APSFuzz determines whether to add the seed to the seed queue based on its diversity score (line 16-17) and updates the scores and energy of all scenarios (line 18). Finally, all the scenarios that have been run in the simulator are added to tested_scenarios (line 19). Once the list reaches its size limit, the fuzzing process for a new seed will begin.

3 Evaluation

3.1 Evaluation Setup

Testing Platform. We chose Autoware.Universe [4] as our test target. Autoware is a well-known open-sourced high-level ADS that includes a well-functioning automatic parking module. All experiments were conducted using version 0.9.13 of the CARLA simulator [10].

Prototype. We implemented a prototype of APSFuzz with 5K+ lines of Python code. Carla Scenario Runner [8] was used to parse openscenario [3]. In our NPC planner module, we reuse the hybrid A* algorithm ROS [21] node of Autoware [5] to plan parking trajectories for dynamic vehicles.

Scenario Dataset. The maps used in our experiments are categorized into three types based on the parking style: parallel parking, perpendicular parking, and angle parking. For each map, we manually constructed several initial seed scenarios (only contains EGO's $\mathbb{IP}$ and $\mathbb{GP}$).

Table 2: Results of ablation studies.

Map	Method	Diverse Buggy Scenario				
		Collision	Stuck	Timeout	Pose_error	Sum
parallel_map_1	$APSFuzz$	**0**	**7**	**7**	**3**	**17**
	$APSFuzz_\alpha$	0	1	0	1	2
	$APSFuzz_\beta$	0	5	3	1	9
	$APSFuzz_\gamma$	0	3	0	2	5
parallel_map_2	$APSFuzz$	**1**	**5**	**5**	**2**	**13**
	$APSFuzz_\alpha$	0	3	0	0	3
	$APSFuzz_\beta$	0	**5**	4	0	9
	$APSFuzz_\gamma$	1	4	0	0	5
angled_map_1	$APSFuzz$	**1**	**6**	**1**	**4**	**12**
	$APSFuzz_\alpha$	0	0	0	1	1
	$APSFuzz_\beta$	0	3	1	**4**	8
	$APSFuzz_\gamma$	0	3	0	2	5
angled_map_2	$APSFuzz$	**3**	**5**	1	**3**	**12**
	$APSFuzz_\alpha$	0	1	0	0	1
	$APSFuzz_\beta$	1	4	**2**	2	9
	$APSFuzz_\gamma$	0	4	1	3	8
perpendicular_map_1	$APSFuzz$	**1**	**4**	3	1	**9**
	$APSFuzz_\alpha$	0	1	0	**2**	3
	$APSFuzz_\beta$	0	0	**4**	1	5
	$APSFuzz_\gamma$	0	3	3	1	7
perpendicular_map_2	$APSFuzz$	**2**	**5**	2	**2**	**11**
	$APSFuzz_\alpha$	0	1	0	0	1
	$APSFuzz_\beta$	0	1	1	1	3
	$APSFuzz_\gamma$	0	4	**5**	1	10
Sum of 6 maps	$APSFuzz$	**8**	**32**	**19**	**15**	**74**
	$APSFuzz_\alpha$	0	7	0	4	11
	$APSFuzz_\beta$	1	18	15	9	43
	$APSFuzz_\gamma$	1	21	9	9	40

Experiment Environment. Our experiments were conducted on a server running Ubuntu 20.04, equipped with 48 Intel Xeon Gold 6342 CPUs and an NVIDIA GeForce RTX 4090 GPU. This hardware configuration is sufficient to meet the demands of our experimental requirements.

Baseline Tool. To the best of our knowledge, APSFuzz is the first to apply fuzzing-based simulation testing to APS rather than ADS. Moreover, existing ADS fuzzers [9, 15, 17–20, 23, 24, 26, 27] typically rely on lane line information to generate and mutate scenarios (e.g., to adjust lane-changing/lane-following maneuvers). However, these lane-based methods are not applicable for parking scenarios due to the absence of lane lines. Hence, we did not compare APSFuzz with existing fuzzers but chose to design ablation experiments to understand the advantages of our design choices.

3.2 Ablation Experiments

Experiment Design. We design an ablation study to investigate the usefulness of each key-technique of our tool. First, we exclude the Offline NPC Trajectory Generation (denoted as $APSFuzz_\alpha$). We set the NPC's parameters randomly and the generated trajectories might not intersect with EGO's. Second, we exclude the Runtime NPC Trajectory Adjustment (denoted as $APSFuzz_\beta$). The NPCs in the scene will drive at a randomly preset speed. Third, we exclude the Multi-objective Seed Scheduling (denoted as $APSFuzz_\gamma$). The seed scheduling method is replaced by random selection. For each map, we select the same initial seeds. Each seed is used to run 80 scenarios. The effectiveness of our tool is evaluated primarily

Table 3: Identifed root causes on Autoware.

ID	Module	Description	Status
R1	Planning	When the vehicle slightly deviates from the planned parking path, it may collide with surrounding obstacles	○
R2	Planning	Overly conservative security considerations when calculating bounding boxes of obstacles	◐
R3	Planning	Autoware will terminate the automated parking prematurely due to improper arrival check	◐
R4	Planning	Inappropriate parking path search implementation causing frequent replanning	●
R5	Planning	Inability of the prediction and planning modules to accurately identify the driving intentions of the NPC	○

○ indicates not confirmed yet; ◐ indicates confirmed but not patched; ● indicates confirmed and patched.

by counting the number of unique buggy scenarios detected during fuzzing. Moreover, following conventional evaluation practices [17, 19], we also considered other metrics, including (1) the average scenario execution time, (2) the convergence time to discover unique buggy scenarios and (3) the average time to identify the first misbehavior scenario, to faithfully demonstrate the effectiveness and efficiency of APSFuzz.

Experiment Results. As shown in Table 2, APSFuzz discovered the most diverse buggy scenarios across six maps. Compared to the three control groups in ablation experiments, APSFuzz increased the average number of misbehaviors discovered across six maps by 85%, 41%, and 45%. Regarding effectiveness and efficiency, across all maps, APSFuzz requires an average of 3.35 minutes to execute a scenario (including mutation and scheduling), converges to the discovery of unique bugs within an average of 236.1 minutes, and identifies the first misbehavior scenario within an average of 13.5 minutes.

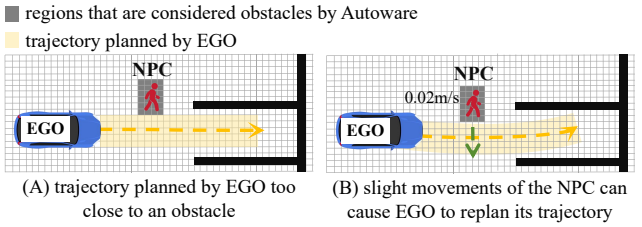
Root Cause Analysis. After manual analysis, we categorized the bugs discovered by APSFuzz into five root causes. So far, one of them has been fixed, and two have been confirmed by the developers. Here, we provide a brief introduction (shown in Table 3).

3.3 Case Study

We present a more detailed analysis of *Root Cause 4* (see Table 3). Autoware uses the Hybrid A* algorithm to plan parking trajectories. During the path search process, this algorithm incorporates various factors into the cost calculation, ultimately finding the path with the minimum cost. However, since Autoware only considers the impact of distance and reverse when calculating costs (shown in Listing 1, line 4 and 9), the planned trajectory may be too close to an obstacle, as long as the obstacle does not obstruct the trajectory (shown in Figure 3 (A)). As a result, even a slight movement of the NPC can cause EGO to perceive a new obstacle on the path, prompting it to stop and re-execute the path planning process. When an NPC is approaching EGO's planned parking trajectory at an extremely slow speed (e.g., 0.02m/s), it will force EGO to continuously replan its trajectory (shown in Figure 3 (B)) which ultimately leads to a timeout.

Listing 1 Faulty Code of Case.

```
1 double AstarSearch::estimateCost(...){...
2     total_cost +=
3       calcReedsSheppDistance(...) *  distance_heuristic_weight;...}
4 bool AstarSearch::search(){...
5     const double move_cost = is_turning_point
6         ? planner_common_param_.reverse_weight * transition.distance
7         : transition.distance;...}
```



(A) trajectory planned by EGO too close to an obstacle

(B) slight movements of the NPC can cause EGO to replan its trajectory

Figure 3: Parking timeout caused by frequent trajectory replanning.

4 Related Work

Dataset-Based APS simulation Testing. Some existing works have constructed datasets of parking scenarios [16, 22] based on real-world traffic flow in parking lots for simulation testing. However, the scenes in these datasets are limited, unable to cover a sufficient variety of situations, and they cannot be mutated to generate more challenging scenarios.

Simulation-Based ADS Fuzzing Testing. Current existing works primarily focus on scenario testing in public road scenarios [9, 15, 17–20, 23, 24, 26, 27]. They require lane information to generate and mutate behavior-specific trajectories for NPCs (e.g., for lane-change maneuvers). However, these lane-based approaches are not applicable for parking scenarios, which typically occur in free spaces without lane markings.

5 Limitations

Parking Maps. Although our current maps include three types of parking spaces. These maps do not account for those involving ramps or spiral ramps. Additionally, the current parking space layout is relatively orderly and does not cover cases where the distribution of parking spaces is scattered or complex in certain scenarios. In the future, we plan to test

APSFuzz in more diverse parking maps.

Testing Environment. Autoware's detection and localization module suffers from the issues [13, 14] of low point cloud recognition accuracy for pedestrians and long localization time when connected with Carla. To tackle these issues and ensure convincing evaluation results, we here follow BehAVExplor's [9] strategy, i.e., to dynamically inject the ground-truth perception and localization data to the target APS. Apparently, this strategy would inevitably cause APSFuzz to overlook potential safety flaws within perception and localization components. That said, our evaluation results have demonstrated the usefulness of APSFuzz, at least in uncovering safety-critical bugs in the remaining APS components.

Real-world Validation. In this work, we merely conducted simulation testing, rather than real-world physical testing, to identify and confirm the existence of safety-critical bugs in the target APS. Hence, it would inevitably cause concerns about the realism of identified bugs. However, considering that the identified bugs have been confirmed or patched by APS developers (detailed in Table 3), we believe our simulation-based methodology can indeed improve the safety levels of APSs. In the future, we will try our best to set up a physical experimental vehicle equipped with target APS to help confirm bugs in a more convincing way.

6 Conclusion

In this work, we introduce APSFuzz, a simulation-based fuzzer designed for detecting software bugs of APSs. (1) We first define the NPC behavior modeling and testing oracles in parking scenarios. (2) Then, based on EGO trajectory of the previous execution, we propose an interaction-oriented mutation strategy to enhance the effectiveness of mutation. (3) Finally, we introduce a multi-objective seed scheduling mechanism to further improve the ability to discover diverse buggy scenarios. Evaluation on Autoware demonstrated the effectiveness and efficiency of APSFuzz.

7 Acknowledgment

We would like to thank the anonymous reviewers for their insightful comments that helped improve the quality of the paper. This work was supported in part by the National Natural Science Foundation of China (62402116). Min Yang is the corresponding author, and a faculty of Shanghai Institute of Intelligent Electronics & Systems, and Engineering Research Center of Cyber Security Auditing and Monitoring, Ministry of Education, China.

References

[1] Global Automated Valet Parking (AVP) Market Analysis: Size, Share, Growth, Trends, Challenges, and Opportunities (2024-2030). https://globalinsightspartner.pragmainsights.com/reports/1035/automated-valet-parking-market, 2024. [Online; accessed Feb 2025].

[2] What's REALLY Causing Tesla FSD v12.5.4 AutoPark Failures? https://www.youtube.com/watch?v=uKDIfc-X0Pw, 2024. [Online; accessed Feb 2025].

[3] ASAM e.V. ASAM OpenSCENARIO®. https://www.asam.net/standards/detail/openscenario, 2023. [Online; accessed Feb 2025].

[4] Autoware Foundation. Autoware. https://autoware.org/, 2024. [Online; accessed May 2024].

[5] Autoware Foundation. Freespace Planner. https://autowarefoundation.github.io/autoware.universe/main/planning/freespace_planner/, 2024. [Online; accessed 2024].

[6] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. Testing vision-based control systems using learnable evolutionary algorithms. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 1016–1026, 2018.

[7] Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, AAAIWS'94, page 359–370. AAAI Press, 1994.

[8] CARLA. ScenarioRunner. https://scenariorunner.readthedocs.io/en/latest/, 2024. [Online; accessed May 2024].

[9] Mingfei Cheng, Yuan Zhou, and Xiaofei Xie. Behavexplor: Behavior diversity guided testing for autonomous driving systems. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2023, page 488–500, New York, NY, USA, 2023. Association for Computing Machinery.

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

[11] International Organization for Standardization. Iso 16787:2017 - intelligent transport systems-assisted parking system (aps) - performance requirements and test procedures. https://www.iso.org/standard/73768.html, 2017. [Accessed: Feb 2025].

[12] International Organization for Standardization. Iso 21448:2022 - road vehicles — safety of the intended

functionality. https://www.iso.org/standard/77490.html, 2022. [Accessed: Feb 2025].

[13] Autoware Foundation. Autoware.universe discussion 4788. https://github.com/orgs/autowarefoundation/discussions/4788, 2023. [Online; accessed Feb 2025].

[14] Autoware Foundation. Autoware.universe issue 4386. https://github.com/autowarefoundation/autoware.universe/issues/4386, 2023. [Online; accessed Feb 2025].

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

[16] Jiawei Hou, Qi Chen, Yurong Cheng, Guang Chen, Xiangyang Xue, Taiping Zeng, and Jian Pu. Sups: A simulated underground parking scenario dataset for autonomous driving. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 2265–2271. IEEE, 2022.

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

[18] Changwen Li, Chih-Hong Cheng, Tiantian Sun, Yuhang Chen, and Rongjie Yan. Comopt: Combination and optimization for testing autonomous driving systems. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 7738–7744. IEEE, 2022.

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

[20] Zhongrui Li, Jiarun Dai, Zongan Huang, Nianhao You, Yuan Zhang, and Min Yang. Viohawk: Detecting traffic violations of autonomous driving systems through criticality-guided simulation testing. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 844–855, 2024.

[21] ROS. Robot Operating System. https://www.ros.org/, 2024. [Online; accessed May 2024].

[22] Xu Shen, Ivo Batkovic, Vijay Govindarajan, Paolo Falcone, Trevor Darrell, and Francesco Borrelli. Parkpredict: Motion and intent prediction of vehicles in parking lots. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1170–1175, 2020.

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

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

[25] Lu Xiong, Jie Gao, Zhiqiang Fu, and Kui Xiao. Path planning for automatic parking based on improved hybrid a* algorithm. In *2021 5th CAA International Conference on Vehicular Control and Intelligence (CVCI)*, pages 1–5, 2021.

[26] Xiaodong Zhang, Wei Zhao, Yang Sun, Jun Sun, Yulong Shen, Xuewen Dong, and Zijiang Yang. Testing automated driving systems by breaking many laws efficiently. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2023, page 942–953, New York, NY, USA, 2023. Association for Computing Machinery.

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