VioHawk: Detecting Traffic Violations of Autonomous Driving Systems through Criticality-guided Simulation Testing

ABSTRACT

As highlighted in authoritative standards (e.g., ISO21448), traffic law compliance is a fundamental prerequisite for the commercialization of autonomous driving systems (ADS). Hence, manufacturers are in severe need of techniques to detect harsh driving situations in which the target ADS would violate traffic laws. To achieve this goal, existing works commonly resort to searching-based simulation testing, which continuously adjusts the scenario configurations (e.g., add new vehicles) of initial simulation scenarios and hunts for critical scenarios. Specifically, they apply pre-defined heuristics on each mutated scenario to approximate the likelihood of triggering ADS traffic violations, and accordingly perform searching scheduling. However, with those comparably more critical scenarios in hand, they fail to offer deterministic guidance on which and how scenario configurations should be further mutated to reliably trigger the target ADS misbehaviors. Hence, they inevitably suffer from meaningless efforts to traverse the huge scenario search space.

In this work, we propose VioHawk, a novel simulation-based fuzzer that hunts for scenarios that imply ADS traffic violations. Our key idea is that, traffic law regulations can be formally modeled as hazardous/non-hazardous driving areas on the map at each timestamp during ADS simulation testing (e.g., when the traffic light is red, the intersection is marked as hazardous areas). Following this idea, VioHawk works by inducing the autonomous vehicle to drive into the law-specified hazardous areas with deterministic mutation operations. We evaluated the effectiveness of VioHawk in testing industry-grade ADS (i.e., Apollo). We constructed a benchmark dataset that contains 42 ADS violation scenarios against real-world traffic laws. Compared to existing tools, VioHawk can reproduce 3.1X~13.3X more violations within the same time budget, and save 1.6X~8.9X the reproduction time for those identified violations. Finally, with the help of VioHawk, we identified 9+8 previously unknown violations of real-world traffic laws on Apollo 7.0/8.0.

KEYWORDS

autonomous driving system, traffic violation, simulation testing

1 INTRODUCTION

Autonomous driving systems (ADS, see §2.1) represent a groundbreaking evolution in the realm of transportation, promising heightened mobility and convenience. Nowadays, manufacturers [7] are fervently investing in the development of autonomous vehicles (AV) and are eager to deploy them on public roads. However, safety has obviously become a major obstacle to the widespread deployment of ADSs, considering the fact that AV-involved traffic accidents or violations continue to occur [4]. To tackle these safety issues, manufacturers [7] urgently require advanced testing methodologies to detect possible ADS misbehaviors in various driving situations.

In this state of confusion, virtual simulation testing, due to its flexibility and low cost for curating driving environments with any desired characteristics (e.g., weather conditions, traffic patterns, etc.), has been embraced as a fundamental approach to testing ADSs. Built on top of ADS simulation platform [24, 46] (see §2.2), we observe that existing works [23, 27, 28, 33, 42, 45, 57, 59] mostly put emphasis on detecting general ADS accidents (e.g., collision and out-of-road). However, in contrast, less light has been shed on verifying the traffic law compliance of ADS driving behaviors, which is highlighted in international/regional standards (e.g., ISO 21448 [8]) as an essential safety prerequisite for the commercialization of ADS.

In practice, traffic laws usually encompass a wide range of rules governing diverse driving intentions (e.g., turning, parking, etc.) and could vary depending on specific driving conditions (e.g., locations, traffic flows, etc.). Compared to the detection of general ADS accidents, this complex nature of traffic laws poses greater challenges in curating scenarios in which the target ADS would likely break given laws under law-specified driving conditions.

To address the above challenge, existing works [31, 32, 37, 51, 56] (see §2.3) suggest using searching-based (or fuzzing-based) simulation testing to test ADSs [3, 5, 6] against traffic laws. Generally, given initial scenarios as inputs, they work by iteratively mutating the scenario configurations (e.g., adding a new traffic actor), so as to identify previously unknown violation scenarios by traversing the whole scenario search space. Typically, to navigate the searching process towards violation scenarios of target laws, they enhance the searching scheduling mechanism with pre-defined fitness functions. To be specific, given a mutated simulation scenario, they would either utilize pre-execution AI-based prediction or post-execution analysis to quantify its likelihood of triggering target ADS violations. For example, if post-execution analysis finds that the maximum runtime speed of ADS is 60 km/h in this scenario, it is considered to have a 0.75 (=60/80) likelihood score to break the speed limit of 80km/h. Subsequently, combined with advanced searching algorithms (e.g., evolutionary algorithm), mutation resources in next rounds of searching would be focused on those scenarios with critical likelihood scores.

Key Observation. Here, we observe that, although the aforementioned methodologies [31, 32, 37, 51, 56] can help focus mutation resources on comparably more critical scenarios during scenario searching, they cannot provide deterministic guidance about exactly which and how scenario configurations should be further mutated to trigger target violations. The key reason is that, they merely use discrete indicators (e.g., AV's maximum speed, distances between AV and other vehicles, etc.) to score the criticality of a mutated scenario, which significantly approximate the scenario semantics. Unlike conventional software testing that has high throughput (e.g., hundreds of executions per second [2]), ADS simulation testing could take tens of seconds [57] to execute only one scenario. Consequently, as proved in §5, with unfocused mutations applied on numerous scenario configurations, existing works [37, 51] inevitably suffer from extensive exploration of the vast scenario search space, consequently showing limited effectiveness in exposing ADS traffic violations.

Our Work. In this work, we propose VioHawk, a novel simulation-based ADS fuzzer, to address the limitations mentioned above. Our key insight hints that, traffic laws, especially those that regulate vehicle driving maneuvers (e.g., brake, throttle, turn, etc., scope of considered laws clarified in §2.3), can be unambiguously formalized by classifying all physically reachable areas on the map into **hazardous** and **non-hazardous** areas. For example, when the traffic light is red, areas near the stop line are considered hazardous. As such, violating a traffic law is equivalent to driving into hazardous areas specified by the law. Owing to this insight, the key advantages of VioHawk are twofold:

- *Reliable Quantification of Scenario Criticality for Fuzzing Scheduling.* At any instant of time in a given simulation scenario, we can leverage Vehicle Dynamics Model (VDM) to predict the drivable areas [49] of AV in near future (i.e., areas that AV can potentially reach without violating vehicle dynamics or colliding with obstacles). After calculating the intersection of AV drivable areas and law-specified hazardous areas, we can faithfully quantify the possibility for ADS to break traffic laws in this scenario (i.e., to drive into law-specified hazardous areas). This criticality quantification mechanism can greatly ease the fuzzing scheduling (i.e., favor scenarios with high criticality).
- *Deterministic Guidance on Scenario Mutation Operations.* Considering the above, an appealing solution to trigger ADS traffic violations is to mutate the scenario configurations (e.g., add new traffic actors or obstacles, adjust driving maneuvers of surrounding vehicles) to induce the target AV to drive into those hazardous areas. For example, if the law-specified hazardous area is located in the left front of the target AV, we obstruct the drivable areas to the right and front as extensively as possible.

Although the aforementioned idea sounds straightforward, as thoroughly discussed in §3, we overcome non-trivial technical challenges to ensure the usability and effectiveness of VioHawk.

Evaluation Results. We conducted extensive experiments to demonstrate the effectiveness of VioHawk. Here, we chose Apollo 7.0 [3] (i.e., the stable version of an industry-grade high-level ADS) as our evaluation target. Since there are no publicly available datasets of simulation scenarios (i.e., scenario configuration files that are compatible with the ADS simulation platforms [24, 46]) that describe ADS traffic violations, we tried our best to construct such a dataset from scratch. Specifically, based on the videos [51] and textual reports [55] of ADS traffic violations published by existing works, we successfully reproduced 42 simulation scenarios in which Apollo 7.0 would break specific U.S./Chinese traffic laws.

In the evaluation, we devoted extensive efforts to thoroughly compare VioHawk with six related works in the SE/Security community that hunt for ADS traffic violations or general ADS accidents, namely Lawbreaker [51], ABLE [56], SAMOTA [31], AutoFuzz [57], DriveFuzz [37] and AV-Fuzzer [42]. To achieve a fair comparison, these tools are fed with the same initial simulation scenarios to identify possible violation scenarios through scenario mutation (i.e., 3 hours allowed to mutate each initial scenario). Finally, the results show that VioHawk successfully reproduced 40/42 violation scenarios (95.2%), while the baseline tools only reproduced no more than 13/42 violation scenarios (31.0%). Moreover, for the violations successfully reproduced by both VioHawk and baseline tools, VioHawk could save 1.6X~8.9X the reproduction time.

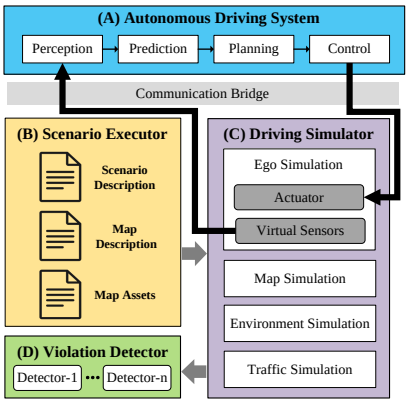


Figure 1: Architecture of ADS Simulation Platform.

Additionally, to further demonstrate the utility of VioHawk, we considered another 10 Chinese and U.S. traffic laws and identified 9/10 previously unknown traffic violations of Apollo 7.0, and 8/10 violations of Apollo 8.0. Through careful case studies, we confirmed the code-level root causes of identified violations, which can bring insights about how to mitigate these safety-critical issues.

Contributions. In summary, we make the following contributions:

- We introduce a formal methodology that unambiguously describes traffic law regulation via hazardous driving areas.
- We propose VioHawk, a simulation-based ADS fuzzer that hunts for ADS traffic violations by inducing AV to drive into law-specified hazardous areas.
- We conducted extensive experiments to demonstrate the effectiveness of VioHawk when applied to industry-grade ADS.
- We open-source our benchmark dataset, as well as the prototype of VioHawk, to facilitate follow-up research (see §9).

2 BACKGROUND

2.1 Autonomous Driving System (ADS)

Architecture of High-level ADS. As illustrated in Figure 1(A), high-level ADSs (e.g., Baidu Apollo [3]) typically employ a multi-module architecture, consisting of the following four main functional modules. ❶ **Perception** receives sensor data (e.g., camera, LiDAR, etc) and senses the environment through the recognition of surrounding obstacles and the identification of traffic signals (e.g., the status of traffic lights); ❷ **Prediction** anticipates the future trajectories of the surrounding obstacles; ❸ **Planning** is responsible for decision-making regarding desired driving behaviors of EGO (i.e., target AV). It utilizes information from road maps and traffic conditions (e.g., surrounding vehicles and traffic signals) to generate a trajectory that is both safe and compliant with traffic laws. ❹ **Control** generates vehicle control commands (e.g., throttle, brake, and steering) to direct EGO in adhering to the planned trajectory.

2.2 ADS Simulation Testing

Architecture of ADS Simulation Platform. As shown in Figure 1, the architecture of the ADS simulation platform involves three key

components. ❶ **Scenario Executor** is responsible for loading the simulation scenario, which describes environmental conditions (e.g., map roads, weather conditions, and, etc) as well as dynamic traffic conditions (e.g., maneuvers of EGO and NPC vehicles), into the driving simulator. ❷ **Driving Simulator** simulates the driving scenario to test the target ADS. Upon finishing the test, ❸ **Violation Detector** assesses whether a violation (e.g., collision, violation of traffic laws, etc) occurs by verifying the execution results (e.g., the trajectories of EGO and NPC vehicles) with pre-defined oracles.

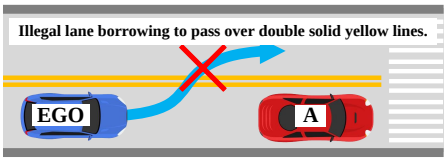
2.3 ADS Traffic Violations

Target Scope of Traffic Laws. Traffic laws enforce a set of regulations to govern the behaviors and interactions of different road participants (e.g., vehicles and pedestrians). In this work, we primarily focus on the traffic laws that constrain driving maneuvers of vehicles (e.g., parking, braking, turning, overtaking, etc.), while other laws including those that govern the driving signals (e.g., the turn signal) are currently considered out of scope.

Example of ADS Traffic Violation. Referring to the traffic laws of New York [11], EGO should not change lanes across double yellow lines. However, in the scenario depicted in Figure 2(A), EGO equipped with Apollo 7.0 makes an illegal overtaking decision on the road with double yellow lines, thus violating this law. As illustrated in Figure 2(B), this violation arises from an implementation deficiency that when assessing the road conditions of lane changes, Apollo only considers single solid lines as the prohibiting condition, while overlooking double yellow lines.

Existing Works. LawBreaker [51] and ABLE [56], the two most relevant state-of-the-art works, leverage simulation-based fuzzing to detect ADS violations of various real-world traffic laws. LawBreaker [51] proposes a rich domain-specific language (DSL) to encode real-world traffic laws. The DSL helps implement the violation oracles to observe the occurrence of target violations. It also helps quantify the likelihood of EGO violating traffic laws as a criticality score and randomly mutate seeds with high scores. Built on top of the DSL of LawBreaker [51], ABLE [56] enhances an existing model called GFlowNet [20] to generate diverse testing scenarios that can trigger violations of traffic laws. It first defines rewards based on how close a scenario is to violating a traffic law, using the robustness semantics of signal temporal logic. Then, it uses active learning to dynamically update the rewards, focusing on uncovered violations and favoring the exploration of new scenario spaces. However, these two works all lack deterministic guidance for mutations, making it hard to generate violation scenarios with carefully constructed NPCs (e.g., see Figure 2, to deploy NPC-A in specific positions to strategically induce EGO to change lanes).

Some works [31, 32, 37], although proposed with the capability to detect ADS traffic violations, can currently handle only simple traffic laws (i.e., speed limits [37] and traffic lights [31, 32]). DriveFuzz [37] assesses EGO's driving quality by recording the observed dangerous driving behaviors (e.g., hard acceleration) during scenario execution, and preferentially applies random mutation operators on scenarios with lower driving quality. Both MORLOT [32] and SAMOTA [31] utilize the many-objective searching strategy for generating scenarios that break predefined safety requirements (e.g., to comply with traffic lights). Specifically,



(A) ADS Traffic Violation

```
void PathLaneBorrowDecider::CheckLaneBorrow(·····)
  ·····
291     if (*left_neighbor_lane_borrowable) {
292         lane_boundary_type = LeftBoundaryType(waypoint);
293         if (lane_boundary_type == SOLID_YELLOW ||
294 +           lane_boundary_type == DOUBLE_YELLOW ||
295             lane_boundary_type == SOLID_WHITE) {
296           *left_neighbor_lane_borrowable = false;
  ·····
```

(B) Root Cause and Patch of the Traffic Violation

Figure 2: Example of Traffic Violation in Apollo 7.0.

to improve the effectiveness and efficiency of scenario searching, SAMOTA [31] enhances existing many-object searching algorithms with surrogate-assisted optimization, and MORLOT [32] combines multi-objective searching with Reinforcement Learning. Similar to LawBreaker [51] and ABLE [56], without deterministic guidance on the selection of mutation operators, these three works [31, 32, 37] also suffer from extensive but valueless exploration of the vast scenario search space. Besides, other works [23, 27, 28, 30, 33, 40, 41, 43, 48, 52, 53, 57] focus primarily on the detection of general ADS accidents (e.g., collision and out-of-road), and cannot handle complex traffic laws. There is another line of work, AVChecker [55], that uses static analysis to detect violations in ADS code. But its transferability is limited due to the required source code modeling.

3 OVERVIEW

3.1 Overall Idea

Key Idea. In this work, we are motivated to explore a more effective solution for quantifying the scenario criticality, the results of which can not only provide deterministic guidance on the scenario mutation, but also ease the fuzzing scheduling. Here, our key insight is that, we can formalize the traffic law regulations as **hazardous** areas on the map, and a traffic violation equals driving into **hazardous** areas. Following this idea, the criticality can be faithfully quantified by measuring the proximity between EGO and the law-specified **hazardous** areas. Also, the scenario mutation has a clear guidance, that is to induce EGO towards the **hazardous** areas. In the following subsection (§3.2), we thoroughly discuss the challenges of implementing this idea, as well as our solutions.

3.2 Challenges and Key Techniques

Challenge-I: How to model complex and diverse traffic laws with hazardous areas? The complexity and diversity of traffic laws pose great challenges for our area-based formalization. Traffic laws encompass restrictions on different driving maneuvers, interactions between road participants, etc. The area-based formalization should express these restrictions correctly. Furthermore, traffic laws could vary across regions and change over time. It is not feasible to manually customize the formalization methodology for each traffic law in a one-by-one fashion.

Key Technique-I: Decomposition-based solution for Traffic Law Formalization. Although traffic law regulations are complex and diverse, our key observation hints that, each traffic law (scope

clarified in §2.3) can be decomposed into two parts, including the governed "driving intention" and the declared "driving condition". ❶ *Driving intention* delineates the basic driving maneuvers under regulation at specific road structures (e.g., crossing an intersection), based on which we can determine the superset (e.g., all regions near the intersection) of law-specified hazardous areas. ❷ *Driving condition* enforces the detailed constraints that EGO should satisfy (e.g., do not drive through the stop line) when performing the basic driving maneuvers. It can be expressed by fine-grained restricted areas (i.e., areas where traffic violations could occur, e.g., areas in front of the stop line). Notably, these restricted areas become hazardous areas in specific states (e.g., the traffic light is red). ❸ By calculating the intersection between the areas modeled by *Driving Intention* and the hazardous areas modeled by *Driving Condition*, we can finally obtain law-specified hazardous areas.

Following the above decomposition scheme, we can thus design independent formalization operators, respectively, for describing each possible "driving intention" and "driving condition". With these basic operators in hand, given a previously unknown traffic law, we can achieve the area-based formalization by combining pre-defined operators, rather than constructing it from scratch.

Challenge-II: How to utilize law-specified hazardous areas to guide simulation-based fuzzing? As briefly introduced in §3.1, given the law-specified hazardous areas on the map, we seek to induce EGO to drive into the hazardous areas by mutating the scenario configurations (e.g., add new NPC vehicles, adjust driving maneuvers of NPC vehicles, etc.). However, the induction is technically a non-trivial task. It is mainly because ADS is a complex non-deterministic system, containing multiple probabilistic-based AI models and decision algorithms. Theoretically, without actually executing the mutated simulation scenario, it is hard to predict what kind of mutation operations can help achieve the induction.

Key Technique-II: Criticality-based Seed Scheduling and Criticality-guided Mutation. To boost the effectiveness of fuzzing, we first introduce a scenario scheduling mechanism to focus the mutation resources on the most critical ones. Specifically, given a mutated scenario, we employ a classical kinetic model to compute EGO's drivable areas (i.e., areas that are reachable merely according to the vehicle dynamics) at each timestamp during runtime. After that, we can faithfully quantify the scenario criticality, by calculating the intersection degree between EGO's drivable areas and law-specified hazardous areas. The criticality scores can practically ease the fuzzing scheduling.

As for scenario mutation, although it is infeasible to deterministically force EGO to drive into the hazardous areas, we can implicitly and gradually approach this goal by compressing EGO's non-hazardous drivable areas. In such a manner, EGO would probably have no choice but to drive into the hazardous areas. To be specific, we segment the non-hazardous drivable areas of EGO into 8 sub-partitions, each of which is in a unique direction of EGO. Finally, we design customized mutation operations for compressing the non-hazardous drivable areas in different directions of EGO.

3.3 Workflow of VioHawk

Building upon the aforementioned key techniques, Figure 3 illustrates the general workflow of VioHawk. VioHawk comprises two

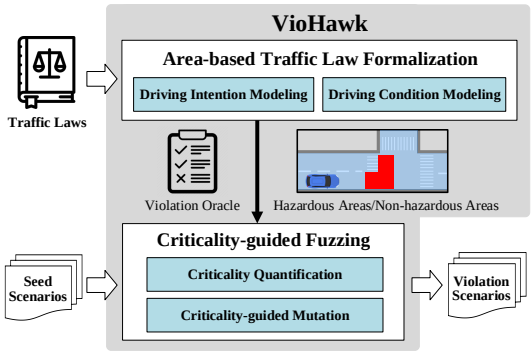


Figure 3: Workflow of VioHawk.

Table 1: The Modeled Driving Intention.

Intention Type	Modeled Areas
Cross Intersection (Go Straight/Turn Left/Turn Right)	$\{\mathbb{I}_i \in \mathbb{MAP}_{\mathbb{I}} \mid \text{distance}(\mathbb{I}_i, \mathbb{P}_{\mathbb{E}}) < 50\}$
Cross Crosswalk	$\{\mathbb{C}_i \in \mathbb{MAP}_{\mathbb{C}} \mid \text{distance}(\mathbb{C}_i, \mathbb{P}_{\mathbb{E}}) < 50\}$
Lane Change (Left/Right)	$\{\mathbb{L}_i \in \mathbb{MAP}_{\mathbb{L}} \mid \text{distance}(\mathbb{L}_i, \mathbb{P}_{\mathbb{E}}) < 50\}$
Lane Follow	$\{\mathbb{L}_i \in \mathbb{MAP}_{\mathbb{L}} \mid \text{distance}(\mathbb{L}_i, \mathbb{P}_{\mathbb{E}}) < 50\}$
Parking	$\mathbb{U}$

essential components, namely *Area-based Traffic Law Formalization* (see §4.1) and *Criticality-guided Fuzzing* (see §4.2, §4.3 and §4.4).

4 APPROACH

4.1 Area-based Traffic Law Formalization

Symbol Definition. We first define necessary symbols to ease the clarification of our formalization methodology. As briefly introduced in §3.2, given a traffic law, the law-specified hazardous areas $\mathbb{A}_H$ are calculated by intersecting the areas $\mathbb{A}_I$ modeled from the governed driving intention and the areas $\mathbb{A}_C$ modeled from the desired driving condition (i.e., including the restricted areas $\mathbb{RA}$ and the corresponding hazardous states $\mathbb{HS}$). To illustrate how we model different driving intentions and driving conditions, we define symbols including $\mathbb{MAP}$ ("map"), $\mathbb{I}$ ("intersection"), $\mathbb{R}$ ("road"), $\mathbb{U}$ ("all road areas"), $\mathbb{L}$ ("lane"), $\mathbb{C}$ ("crosswalk"), and $\mathbb{T}$ ("traffic light"), $\mathbb{DY}$ ("double yellow lines"), and $\mathbb{SL}$ ("stop line") for describing road structures and traffic facilities; For describing traffic actors, we use $\mathbb{E}$ ("EGO vehicle"), $\mathbb{N}$ ("NPC vehicle"), and $\mathbb{PED}$ ("pedestrian"). Lastly, $\mathbb{P}$ and $\mathbb{H}$ refer to the position and heading of a traffic actor.

Driving Intention Modeling (examples shown in Table 1). Driving intention delineates the basic driving maneuvers under regulation at specific road structures. Hence, we can use the map areas of these road structures (i.e., $\mathbb{A}_I$) to model the driving intention in a semantically equivalent manner. To be specific, we follow four steps to finally determine $\mathbb{A}_I$: ❶ We first manually extract the type of road structures (marked as $\mathbb{X}$) described in the text of the traffic law. For example, from "EGO passing the stop line at the intersection", we can determine that the road type is "intersection". ❷ Then, we use map processing tools (e.g., *OSMIUM* [12] for OpenStreetMap [19]) to extract all relevant map areas of our target road structure from the given high-definition(HD) map, i.e., $(\mathbb{MAP}_{\mathbb{X}}=[\mathbb{X}_1,\mathbb{X}_2,...,\mathbb{X}_n])$. ❸ Next, based on the initial states of EGO (i.e., position and orientation) in the simulation scenario s, we

Table 2: The Modeled Driving Condition (Restricted Areas).

Type	Name	Modeled Areas
Areas near Traffic Facilities	In Front of Stop Line	$\text{FrontArea}(\mathbb{SL}_i, 5)$
	On Double Yellow Lines	$\text{Area}(\mathbb{DY}_i)$
	On Crosswalk	$\text{Area}(\mathbb{C}_i)$
	At Intersection	$\text{Area}(\mathbb{I}_i)$
	Near Crosswalk (20 feet)	$\text{NearArea}(\mathbb{C}_i, 20)$
	Near Traffic Light (30 feet)	$\text{NearArea}(\mathbb{T}_i, 30)$
	Near Stop Sign (30 feet)	$\text{NearArea}(\mathbb{S}_i, 30)$
Areas near NPCs	In Front of NPC	$\text{FrontArea}(\mathbb{N}_i, 5)$
	Next to NPC	$\text{LeftArea}(\mathbb{N}_i, 5) \cup \text{RightArea}(\mathbb{N}_i, 5)$

Table 3: The Modeled Driving Condition (Hazardous States).

Type	Name	State Conditions
States of Traffic Actors	EGO's speed is zero	$\forall t' \in [t-\Delta t, t] \quad v_{\mathbb{E}}(t') = 0$
	EGO's heading changes	$\exists t' \in [t-\Delta t, t] \quad (\mathbb{H}_{\mathbb{E}}(t) - \mathbb{H}_{\mathbb{E}}(t') > \theta$
	EGO's position is behind the stop line	$\mathbb{P}_{\mathbb{E}}.\text{isbehind}(\mathbb{SL}_i)$
	NPC's speed is zero	$\forall t' \in [t-\Delta t, t] \quad v_{\mathbb{N}_i}(t') = 0$
	NPC's position is behind the crosswalk	$\mathbb{P}_{\mathbb{N}i}.\text{isbehind}(\mathbb{C}_i)$
	Pedestrian's position is on the crosswalk	$\mathbb{P}_{\mathbb{PED}i} \in \text{Area}(\mathbb{C}_i)$
States of Interaction [1]	EGO and NPC are in adjacent lanes	$\exists \mathbb{L}_i \in \{\mathbb{L}_i \in \mathbb{MAP}_{\mathbb{L}} \mid \mathbb{L}_i.\text{isAdjacent}(\mathbb{P}_{\mathbb{E}})\}$ $\mathbb{P}_{\mathbb{N}i} \in \text{Area}(\mathbb{L}_i)$
	NPC enters intersection before EGO	$\mathbb{P}_{\mathbb{N}i}(t) \in \text{Area}(\mathbb{I}_i) \wedge \mathbb{P}_{\mathbb{E}}(t) \in \text{Area}(\mathbb{I}_i)$ & $\exists t' \in [t-\Delta t, t] \mathbb{P}_{\mathbb{N}i} \in \text{Area}(\mathbb{I}_i) \wedge \mathbb{P}_{\mathbb{E}} \notin \text{Area}(\mathbb{I}_i)$
States of Traffic Lights	Traffic light is red	$\text{Signal}(\mathbb{T}_i) == \text{RED}$
	Traffic light is yellow	$\text{Signal}(\mathbb{T}_i) == \text{Yellow}$

[1] **States of Interaction** refers to the interaction states between EGO and other traffic actors.

filter those areas that are far away (e.g., 50 meters) from the initial position $\mathbb{P}_{\mathbb{E}}$ of EGO vehicle (i.e., these areas can hardly be reached in a limited time of simulation running). Finally, the remaining areas in $\mathbb{MAP}_{\mathbb{X}}$ are grouped and marked as $\mathbb{A}_I$.

$$\mathbb{A}_I(s) = \{\mathbb{X}_i \in \mathbb{MAP}_{\mathbb{X}} \mid \text{distance}(\mathbb{X}_i, \mathbb{P}_{\mathbb{E}}) < 50\}$$

Notably, there exist some traffic laws (e.g., "Parking") that do not specify the road structures regarding the driving intention. In this condition, all map road areas will be extracted as $\mathbb{A}_I(s)$ by default (as shown in Figure 4(C)). In the current version of VioHawk, as shown in Table 1, we have modeled 5 kinds of driving intentions, which are quite common in different traffic laws.

Driving Condition Modeling (examples shown in Table 2 and Table 3). As mentioned in §3.2, we further decompose the driving condition into its restricted areas $\mathbb{RA}$ and a series of hazardous states $\{\mathbb{HS}_1, \mathbb{HS}_2, \ldots, \mathbb{HS}_n\}$ (i.e., state conditions that the restricted areas become hazardous areas). Theoretically, the driving condition can be formalized as a tuple:$(\mathbb{RA}, \{\mathbb{HS}_1, \mathbb{HS}_2, \ldots, \mathbb{HS}_n\})$. However, given that restricted areas and hazardous states (e.g. EGO's speed) may continuously change during the scenario execution, we should further enhance this formalization into a time-sensitive one. To sum up, in a simulation scenario s, hazardous areas $\mathbb{A}_C$ modeled from the driving condition at time t can be expressed as:

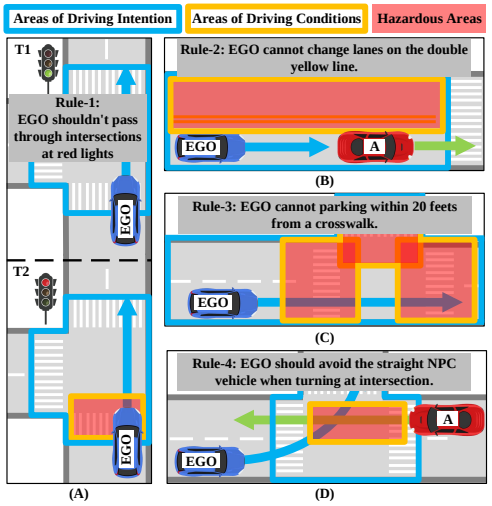


Figure 4: Hazardous Areas of Different Traffic Laws.

$$\mathbb{A}_C(s,t) = \begin{cases} \mathbb{RA}(s,t) & \forall \mathbb{HS}_i(t) == \text{True} \\ \varnothing, & \text{otherwise} \end{cases}$$

The restricted areas can be divided into the following two categories (as shown in Table 2). ❶ **Areas near Traffic Facilities** comprise road areas near or in the locations of traffic facilities (e.g., stopping is not allowed in areas near stop signs); ❷**Areas near NPCs** refer to the dangerous sides relative to NPCs (e.g., when avoiding an NPC in the adjacent lane, the restricted area is the region in front of it).

For the hazardous states, we can model them into three categories (as illustrated in Table 3). ❶ **States of Traffic Actors** refer to the state (e.g., speed, heading, and position) of traffic actors (e.g., EGO, NPCs, and pedestrians) within the scenario. ❷ **States of Interaction** describe the interaction behaviors (e.g., their relative positions) between EGO and other traffic actors. ❸ **States of Traffic Lights** refer to different statuses of traffic lights.

Notably, in particular driving situations, driving conditions should be modeled as multiple separate areas. For instance, for the traffic law specifying "turning vehicles should avoid vehicles going straight", the restricted areas should be those in front of the NPC vehicles. As such, if there exist multiple NPC vehicles in a driving scenario, the driving condition should be modeled as multiple separate areas (i.e., one area for each NPC vehicle):

$$\{\mathbb{A}_{C_1}(s,t), \mathbb{A}_{C_2}(s,t), \ldots, \mathbb{A}_{C_n}(s,t)\}$$

Law-specified Hazardous Area. In a simulation scenario s, based on the formalization results of the driving intention and the driving condition (i.e., $\mathbb{A}_I$(s) and $\mathbb{A}_C(s,t)$), the law-specified hazardous areas at timestamp t can finally be defined as the intersection between $\mathbb{A}_I$(s) and $\mathbb{A}_C(s,t)$:

$$\mathbb{A}_H(s,t) = \bigcup_{i \in \{1,\ldots,n\}} (\mathbb{A}_I(s) \cap \mathbb{A}_{C_i}(s,t))$$

Running Example. Figure 4 illustrates the set of polygonal areas corresponding to driving intentions, driving conditions and the law-specified hazardous areas. Notably, for calculating the intersection

between the polygons, we depended on an established third-party library named Shapely [17], which offers a feature-rich geometry interface for individual geometries and shows high performance on operations with arrays of geometries.

Violation Oracle. Based on the formalization of law-specified hazardous areas $\mathbb{A}_H(s,t)$, we can accordingly develop violation oracles to sense the occurrence of traffic violations. Specifically, based on these oracles, a violation is reported if the EGO vehicle drives into the law-specified hazardous areas at any timestamp during the simulation testing. With the bounding box of EGO marked as $\mathbb{B}_{\mathbb{E}}$, we implement the violation oracle $\rho(s)$ as follows:

$$\rho(s) = \begin{cases} \text{True} & \exists t \in \mathbb{T}, \mathbb{B}_{\mathbb{E}}(t) \cap \mathbb{A}_H(s,t) \neq \varnothing \\ \text{False} & \text{otherwise} \end{cases}$$

4.2 Criticality-guided Fuzzing Algorithm

Based on the formalization results of traffic laws (see §4.1), we propose a novel criticality-guided fuzzing algorithm to identify possible traffic violations. As listed in Algorithm 1, ❶ VioHawk take as inputs a corpus S of initial scenarios and a targeted traffic law R. ❷ In each iteration, VioHawk employs a refined seed scheduling strategy (i.e., roulette wheel selection [29]) to select a scenario from S. Subsequently, a specialized mutation process is applied to the selected scenario to generate more critical scenarios (see §4.4). ❸ The mutated scenario is then executed and the simulation results will be recorded. After verifying the simulation results with the violation oracle, if EGO does violate traffic law R, VioHawk will report this violation. ❹ Each scenario is then evaluated for its criticality regarding proximity to violation (see §4.3) which assigns a score to it. ❺ Finally, the evaluated scenario is re-queued, and the fuzzing cycle continues until either the timeout is reached or an external abort signal is triggered.

Algorithm 1 Fuzzing Algorithm

Input: Seed Corpus S, Traffic Law R
Output: Violation Set V
1: Initial run all seeds in the corpus
2: **repeat**
3: s = chooseNext(S) ▷ Our Seed Scheduling Mechanism
4: s' = mutate(s) ▷ Our Mutation Mechanism
5: Execute s' and obtain trace
6: **if** s' violates R **then**
7: add s' to V
8: $\Phi(s')$ = computeScore(s') ▷ Our Seed Scheduling Mechanism
9: add s' to S
10: **until** timeout reached or abort-signal

4.3 Criticality-based Seed Scheduling

Criticality Quantification. Generally, we try to quantify the criticality (i.e., to calculate the criticality score) of a mutated scenario based on its execution results on the ADS simulation platform. Here, our key idea for criticality quantification (listed in Algorithm 2) is to evaluate the likelihood that EGO drives into the law-specified hazardous areas. Firstly, the scenario would be divided into several time frames based on a pre-defined time interval (i.e., set to 4s). Then, for each selected time frame, we calculate its criticality score by measuring the hazardous drivable areas at this frame, i.e.,

Algorithm 2 Criticality Quantification Algorithm

Input: Scenario s
Output: Criticality Score Φ
1: **function** computeScore(s) ▷ a given scenario that contain multiple frames
2: $\Phi = 0$ ▷ criticality score of this scenario
3: $\mathbf{T}$ = devideIntoTimeFrame(s, 4)
4: **for** each time frame $t \in \mathbf{T}$ **do**
5: $\mathbf{A_H}$ = calcHazardousArea(s, t)
6: $\mathbf{A_D}$ = calcDrivableArea(s, t)
7: $\mathbf{A_O} = \mathbf{A_H} \cap \mathbf{A_D}$ ▷ hazardous drivable areas
8: $\Phi_t = \mathbf{A_O}/\mathbf{A_D}$
9: **if** $\Phi_t > \Phi$ **then**
10: $\Phi = \Phi_t$
11: **return** Φ

the proportion of the overlapped area ($\mathbf{A_O}$) between hazardous areas and drivable areas to the total drivable areas ($\mathbf{A_D}$). Finally, the criticality score of this scenario is determined by identifying the maximum criticality score across all frames.

Drivable Area Calculation. As shown in Figure 5, the drivable areas refer to the region of the road that the EGO can traverse without violating vehicle dynamics or colliding with obstacles. To practically calculate the drivable area of EGO, we leverage the algorithm from [49]. The algorithm focuses on motion planning in dynamic traffic, which is challenging due to the unpredictability of other participants, like overtaking on the left or right. Essentially, it is designed to methodically compute the reachable set of all potential movements for the subject vehicle and different areas (e.g., the red and blue areas in Figure 5) represent a different sequence of high-level decisions. To achieve this, the algorithm employs a simplified vehicle model and a conservative approximation of the reachable set. Hence, the computation of the reachable set is over-approximative, ensuring that no feasible trajectory is missed.

Seed Scheduling Algorithm. For seed scheduling, our preference is to prioritize those with higher criticality scores. However, a blind selection of scenarios solely based on top ratings might stuck in a local optimum, potentially overlooking some promising seeds. To alleviate this concern, we employ a roulette wheel selection [29] in seed scheduling by normalizing the criticality scores of all seeds in the seed queue using the following formula to establish the probability of each seed being selected: $p(i) = \frac{f(i)}{\sum_{j=1}^{n} f(j)}$. Here, $f(i)$ denotes the criticality score of the *i-th* candidate seed, and n represents the number of the seed queue.

Algorithm 3 Scenario Mutation Algorithm

Input: Scenario s, Time Frame t
Output: Mutated Scenario s'
1: **function** Mutate(s,t) ▷ t is the time frame with the highest criticality score across all frames of s
2: $\mathbf{A_S} = 0, \mathbf{P_{max}} = 0$
3: **for** each partition p **do**
4: $\mathbf{A'_S}$ = calcSafeDrivableArea(s, t, p)
5: **if** $\mathbf{A'_S} > \mathbf{A_S}$ **then**
6: $\mathbf{P_{max}} = p$
7: $\mathbf{A_S} = \mathbf{A'_S}$
8: s' = applyOperators(s, $\mathbf{P_{max}}$)
9: **return** s'

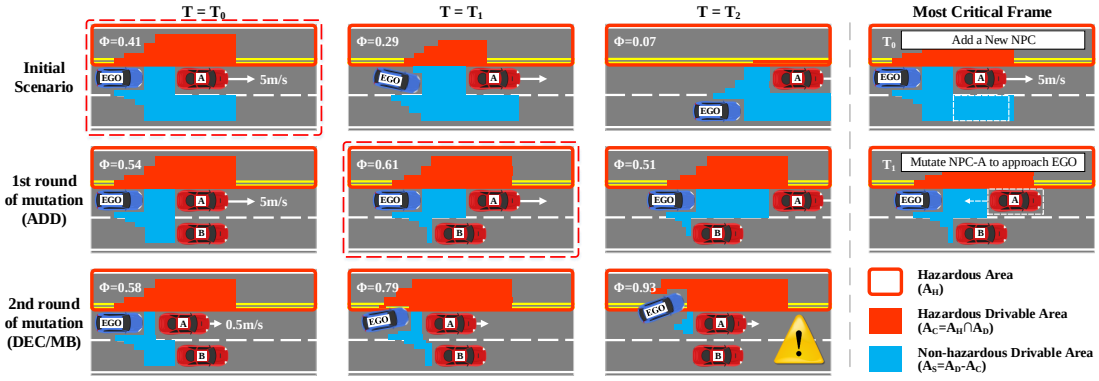


Figure 5: Running Example of Criticality-guided Mutation.

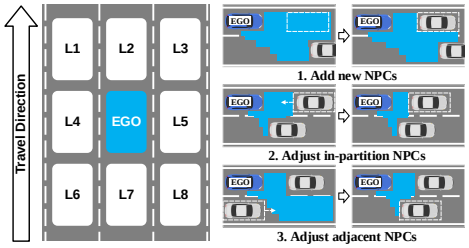


Figure 6: Partition of Surrounding Areas of EGO.

Table 4: The Mutation Operators for Different Partitions.

ID	ADD	Adjust in-partition NPCs		Adjust adjacent NPCs			
		DEC	MB	DEC	MB	ACC	MF
L1	✓	✓	✓			✓	✓
L2	✓	✓	✓				
L3	✓	✓	✓			✓	✓
L4	✓			✓	✓	✓	✓
L5	✓			✓	✓	✓	✓

ADD: Add a new NPC; **DEC**: NPC decelerating; **ACC**: NPC accelerating; **MB**: Moving the NPC backwards; **MF**: Moving the NPC forward.

4.4 Criticality-guided Mutation

Mutation Strategy. Our strategy for scenario mutation is to compress the non-hazardous drivable areas (e.g., the blue areas shown in Figure 5) of EGO, making it have no choice but to drive into the hazardous ones (e.g., the red areas shown in Figure 5). To achieve this goal, VioHawk features a greedy-like mutation algorithm (listed in Algorithm 3). Specifically, given a simulation scenario under mutation, VioHawk first determines its most critical time frame (see §4.3) during the scenario execution. For example, the frame T_0 of the initial scenario shown in Figure 5 is the most critical one when mutating the initial scenario. Then, inspired by ISO-34502 [9] (i.e., principles for scenario-based ADS testing), VioHawk divides the EGO's surrounding areas into 8 zones, labeled as *L1-L8* (shown in Figure 6). Following the division, VioHawk greedily selects the partition with the largest non-hazardous drivable areas to apply specialized mutation operations.

Mutation Operations. To effectively compress the non-hazardous drivable areas within different partitions, as shown in Table 4 and Figure 6, we propose five different mutation operators (detailed as follows). In particular, we do not apply any mutation strategies to L6-L8 since there are no drivable areas behind EGO.

- **ADD.** VioHawk identifies the central point within the partition and deploys a new NPC vehicle at this point. These newly added NPCs would be set to `LaneFollow` mode using the APIs of the simulation platform. They would drive along the lane at a pre-defined speed, with basic capabilities of obstacle avoidance and traffic law compliance (e.g., comply with traffic lights).
- **ACC/DEC.** For NPCs driving at `LaneFollow` mode, VioHawk randomly adjusts its driving speed within a pre-defined reasonable range (i.e., [-5m/s, -1m/s] or [+1m/s, +5m/s]).
- **MF/MB.** VioHawk modifies the initial positions of NPCs with varying chances: a short move (0.5-2m) with a 50% chance, a medium move (2-5m) with a 25% chance, and a long move (5-10m) with a 25% chance. These position changes are made consistent with the road direction based on a curvilinear coordinate system.

Obviously, 3 of the above mutation operators (i.e., ADD/MF/MB) can only be applied at the initial time frame T_0 rather than the most critical time frame T_k. Intuitively, this appears to be contradictory to our greedy-like mutation strategy because these T_0-applied operators might produce unpredictable influences on drivable areas at frame T_k. Here, the rationale behind this design choice is that, there probably exist situations where there are no NPCs in the surrounding areas of EGO at T_k. To ensure the scenario validity, we cannot allow an NPC to suddenly appear at T_k. Consequently, we have no choice but to apply mutation operators at T_0. However, since all newly added NPC vehicles are set to `LaneFollow` mode (i.e., stable driving maneuvers) and the time span of each scenario (e.g., 30s) is quite limited in our problem domain, the positional relationship between the NPC vehicle and EGO at T_0 can be probably maintained until T_k. In such a manner, these T_0-applied mutation operators can also compress the drivable areas at T_k.

Running Example. For clarity, Figure 5 shows how VioHawk mutates an initial scenario to identify a violation of the traffic law that prohibits lane changes crossing double yellow lines.

5 EVALUATION

In this section, we first introduce the experimental setup and the benchmark dataset (§5.1); then we demonstrate the capacity of VioHawk in reproducing known ADS traffic violations (§5.2) and identifying previously unknown ones (§5.3).

Table 5: Collected Traffic Laws in $Dataset_{GT}$.

ID	Data Source	Description	Driving Intention [1]	Driving Condition [1]	
				Restricted Areas	Hazardous States
R-1	[51]&Chinese Law	When the red light is on, vehicles can not cross the intersection.	Cross Intersection	In Front of Stop Line	Traffic light is red.
R-2	[51]&Chinese Law	When the yellow light is on, only vehicles that have crossed the stop line can cross the intersection.	Cross Intersection	In Front of Stop Line	Traffic light is yellow. EGO's position is behind the stop line.
R-3	[51]&Chinese Law	When overtaking, vehicles shall not affect driving of the vehicles in the adjacent lane.	Lane Change	In Front of NPC	EGO and NPC are in adjacent lanes.
R-4	[55]& U.S. Law	When a vehicle has stopped behind the crosswalk, vehicles shall not cross the crosswalk.	Cross Crosswalk	On Crosswalk	NPC's speed is zero. NPC's position is behind the crosswalk.
R-5	[55]& U.S. Law	When a pedestrian is at the crosswalk, vehicles shall not pass the crosswalk.	Cross Crosswalk	On Crosswalk	Pedestrian's position is on the crosswalk.
R-6	[55]& U.S. Law	Vehicles shall not park at the crosswalk.	Parking	On Crosswalk	EGO's speed is zero.
R-7	[55]& U.S. Law	Vehicles shall not park within 20 feet of a crosswalk.	Parking	Near Crosswalk (20 feet)	EGO's speed is zero.
R-8	[55]& U.S. Law	Vehicles shall not park within 30 feet of a traffic light.	Parking	Near Traffic Light (30 feet)	EGO's speed is zero.
R-9	[55]& U.S. Law	Vehicles shall not park within 30 feet of a stop sign.	Parking	Near Stop Sign (30 feet)	EGO's speed is zero.
R-10	[55]& U.S. Law	When a pedestrian is on the crosswalk at the turning side of the intersection, vehicles shall not enter the intersection.	Cross Intersection	On Crosswalk	EGO's heading changes. Pedestrian's position is on the crosswalk.

[1] Decomposition and area-based formalization of traffic laws using VioHawk.

5.1 Experiment Setup

Prototype. We implemented a prototype of VioHawk with 4K+ lines of Python code. Our prototype uses LGSVL Python APIs [1] for scenario mutation and execution, leveraging its user-friendly interfaces to configure scenario attributes and simulate the scenario. Regarding the computation of hazardous areas, a critical functionality of VioHawk, we achieved it by integrating and customizing Commonroad-Reach 2023.1 [36]. According to existing practices [51], we set 30 seconds as the timeout for each scenario execution. Here, we agree that longer execution time of scenarios can help to find more diverse driving situations. However, violations often occur only in specific driving conditions (i.e., road structures and infrastructures, traffic flows, etc.). Therefore, further increasing the execution time of a scenario is not necessarily helpful in our problem domain.

Target ADS. We chose Baidu Apollo [3] as our target ADS, mainly because it is one of the most representative open-source high-level ADSs with widespread commercialization, and has been considered as an evaluation target in various ADS-related researches [21–23, 34, 44, 47, 50–52, 54, 57, 58]. Specifically, we focused on the latest stable version of Apollo (i.e., Apollo 7.0) for the subsequent experiments.

ADS Simulation Platform. The LGSVL [46] is an open-source simulation platform and has gained popularity in ADS simulation testing due to its high fidelity and flexibility. We selected it (i.e., LGSVL 2021.2.2 [10]) as our simulation platform since it offers stable connections with our target ADS (i.e., Apollo). Although the remote services of LGSVL were ceased in January 2022, we have built a local version (see §9) of LGSVL [46] to maintain its usability. Besides, when implementing the prototype of VioHawk, we ensured that the configuration files of the generated simulation scenarios were all compatible with this ADS simulation platform.

Dataset of ADS Traffic Violations: $Dataset_{GT}$. After a systematic literature review, we found that there are no publicly available scenario datasets of ADS traffic violations (i.e., configuration files of simulation scenarios that imply ADS traffic violations). Hence, we decided to construct such a benchmark dataset for our target ADS (i.e., Apollo 7.0) from scratch. Fortunately, existing works [51, 55] in detecting ADS traffic violations have released videos (i.e., screen recording of simulation testing) or textual reports about their identified violation scenarios. Based on these clues, we tried our best to recreate the simulation scenarios (i.e., scenario configuration files) that have equivalent scenario semantics. Specifically, LawBreaker [51] and AVChecker [55] mainly considered U.S./Chinese traffic laws, and identified violations of 27 traffic laws when testing Apollo 5.5/6.0. After over 50 manual hours of investigation, we confirmed that only the violations of 10/27 traffic laws (listed in Table 5) can be successfully reproduced on Apollo 7.0. The reasons for the 17/27 failures are twofold: ❶ **14/17 failures:** The released videos or textual reports cannot provide enough information for the initialization of various scenario configurations (e.g., exact maneuvers of NPCs). ❷ **3/17 failures:** The buggy code pieces responsible for the violations have been patched on Apollo 7.0.

With the 10/27 violation scenarios in hand (i.e., one violation scenario for each traffic law), we tried to further enhance the diversity of our benchmark dataset, so as to ensure convincing evaluation results. Specifically, we traversed the built-in road maps (e.g., San Francisco map [16]) of the simulation platform (i.e., LGSVL), to verify whether these traffic violations can be manually reproduced at different road shapes and structures. Finally, we collected 42 ground-truth violation scenarios (marked as $Dataset_{GT}$) against the 10 traffic laws (listed in Table 5). We believe this benchmark dataset can drive practical value for ADS testing, and the dataset is open-sourced to ease follow-up research (see §9).

Baseline. After careful literature review (see §2.3), we devoted extensive efforts to thoroughly compare VioHawk with 6 baseline tools [31, 37, 42, 51, 56, 57]. Among these baselines, LawBreaker [51] and ABLE [56] are the two most relevant works that can handle complex real-world traffic laws. Furthermore, we also considered AV-Fuzzer [42], DriveFuzz [37], SAMOTA [31] and AutoFuzz [57] as our baselines. They are representative testing tools for detecting general ADS accidents (i.e., collisions and out-of-road) or violations of simple traffic laws (i.e., speed limits and traffic lights). Through these broader comparisons, we can better demonstrate the advantages and necessity of VioHawk. Notably, these baseline tools did not implement a complete set of violation oracles to sense the occurrence of the diverse traffic violations collected in $Dataset_{GT}$. Hence, we enhanced these baseline tools with VioHawk's violation oracles, so as to achieve fair comparison between the effectiveness of violation identification (i.e., LawBreaker [51] and ABLE [56]

Table 6: Traffic Violation Reproduction with VioHawk, LawBreaker, DriveFuzz, AV-Fuzzer, ABLE, AutoFuzz and SAMOTA.

Law ID	GT Violation Scenarios [1]	TTE [2] / # of Seed Queue [3]						
		AV-Fuzzer	DriveFuzz	LawBreaker	ABLE	AutoFuzz	SAMOTA	VioHawk
R-1	(I, C, ↑)	✗/253	✗/164	61m4s /62	23m7s /23	15m49s /16	155m0s /122	8m15s /11
	(I, T, ↑)	✗/264	✗/161	47m4s /46	35m15s /34	27m26s /24	126m41s /112	9m37s /13
	(I, T, ↱)	✗/267	✗/153	✗/203	21m19s /23	✗/157	68m37s /60	13m3s /13
	(I, Y, ↑)	✗/256	✗/161	33m3s /35	21m54s /23	10m32s /11	64m22s /55	11m50s /16
	(I, Y, ↱)	✗/255	✗/167	103m16s /127	18m32s /20	✗/157	36m7s /27	8m57s /12
R-2	(I, C, ↑)	✗/264	✗/168	92m7s /84	102m13s /100	✗/151	✗/145	7m49s /10
	(I, T, ↑)	✗/268	✗/181	64m16s /74	21m32s /25	✗/157	✗/192	13m7s /19
	(I, T, ↱)	✗/267	✗/146	105m16s /120	21m28s /23	✗/157	✗/146	12m43s /18
	(I, Y, ↑)	✗/258	✗/145	78m53s /97	30m3s /34	✗/157	✗/149	13m16s /18
	(I, Y, ↱)	✗/262	✗/171	106m13s /119	25m18s /28	✗/131	✗/134	15m11s /21
R-3	(2L, S, ↑)	✗/265	✗/224	✗/186	✗/147	✗/142	⚠	5m43s /7
	(2L, B, ↑)	✗/238	✗/156	✗/183	✗/186	✗/142	⚠	8m11s /10
	(4L, S, ↑)	✗/274	✗/192	✗/209	✗/154	✗/170	✗/154	129m11s /173
	(4L, B, ↑)	✗/241	✗/164	✗/170	✗/180	✗/167	⚠	✗/193
R-4	(I, C, ↑)	✗/252	✗/159	✗/200	95m36s /110	✗/177	✗/140	11m33s /16
	(I, T, ↑)	✗/272	✗/166	✗/212	20m20s /21	✗/177	159m37s /136	7m24s /10
	(I, T, ↱)	✗/266	✗/147	✗/198	✗/192	✗/210	✗/161	7m23s /10
	(I, Y, ↑)	✗/259	✗/146	✗/209	101m28s /113	✗/138	✗/143	10m19s /14
	(I, Y, ↱)	✗/246	✗/148	✗/205	✗/199	✗/176	✗/146	6m26s /8
R-5	(I, C, ↑)	✗/264	✗/151	✗/182	✗/211	✗/186	✗/151	7m7s /9
	(I, T, ↑)	✗/269	✗/160	✗/183	✗/191	✗/144	✗/148	3m47s /4
	(I, T, ↱)	✗/269	✗/146	✗/177	✗/212	✗/136	✗/152	12m43s /18
	(I, Y, ↑)	✗/260	✗/139	✗/174	✗/199	✗/148	✗/145	5m41s /7
	(I, Y, ↱)	✗/257	✗/152	✗/162	✗/257	✗/150	✗/146	8m31s /11
R-6	(I, C, ↑)	✗/254	✗/163	✗/213	✗/203	✗/173	✗/139	3m49s /4
	(I, T, ↑)	✗/271	✗/164	✗/215	✗/118	✗/210	✗/163	5m42s /7
	(I, T, ↱)	✗/256	✗/145	✗/212	✗/204	✗/170	✗/159	19m48s /29
	(I, Y, ↑)	✗/261	✗/139	✗/208	✗/195	✗/170	✗/158	4m34s /5
	(I, Y, ↱)	✗/246	✗/155	✗/199	✗/216	✗/172	✗/148	8m17s /11
R-7	(I, C, ↑)	✗/257	✗/162	✗/200	✗/230	✗/177	✗/138	11m2s /15
	(I, T, ↑)	✗/272	✗/163	✗/200	✗/207	✗/175	✗/156	4m26s /5
	(I, T, ↱)	✗/264	✗/143	✗/196	✗/195	✗/163	✗/151	5m42s /7
	(I, Y, ↑)	✗/261	✗/141	✗/199	✗/205	✗/177	✗/152	14m55s /19
	(I, Y, ↱)	✗/250	✗/150	✗/185	✗/232	✗/134	✗/139	3m41s /4
R-8	(I, C, ↑)	✗/257	✗/164	✗/202	✗/207	✗/154	✗/138	4m57s /6
	(I, T, ↑)	✗/272	✗/164	✗/207	✗/217	✗/154	✗/164	6m56s /9
	(I, T, ↱)	✗/267	✗/145	✗/236	✗/219	✗/157	✗/154	3m33s /4
	(I, Y, ↑)	✗/261	✗/144	✗/192	✗/206	✗/161	✗/155	3m1s /3
	(I, Y, ↱)	✗/253	✗/153	✗/206	✗/197	✗/133	✗/126	✗/177
R-9	(I, C, ↑)	✗/238	✗/206	✗/168	✗/186	✗/177	✗/141	8m36s /11
R-10	(I, C, ↱)	✗/249	✗/157	✗/188	✗/198	✗/141	✗/145	4m31s /5
	(I, T, ↱)	✗/260	✗/157	✗/194	✗/221	✗/148	✗/146	6m9s /8

[1] **Road Structure(I/2L/4L)**: Intersection/2-Lane/4-Lane; **Road Shape(C/T/Y/B/S)**: Crossroads/T-Junction/Y-Junction/Bend/Straight Road; **Driving Direction(↑ / ↱)**: Going Straight/Turning.

[2] **TTE** records the time required to trigger the traffic violation and the time limit is set to 3 hours.

[3] **# of Seed Queue** refers to the number of scenarios to be executed before triggering the violation or reaching the time limit.

[4] ⚠: scenario cannot be encoded using SAMOTA's vector representation; ✗: violation cannot be reproduced in 3 hours.

Law	Initial Scenario	Visualization
R-1	**Road Structure:** •Intersection with traffic lights **EGO Task:** •Passing through intersection **NPC Conditions:** •None	

Figure 7: Example of Initial Simulation Scenarios.

directly utilize these oracles to calculate fitness scores of scenarios). In addition, necessary implementation efforts are needed to make these baseline tools comply with our testing environment (e.g., file formats of scenarios, the simulation platform, and the target ADS). We addressed these compatibility issues without modifying the original functionalities of these baseline tools (see §9).

Experiment Environment. We conducted all experiments on a Ubuntu 18.04 server with 314 GB of memory, two Intel Xeon Gold 5215 CPUs, and four NVIDIA GeForce RTX 2080 TIs. This hardware configuration provided sufficient stability to run our target ADS and the simulation platform.

5.2 Reproducing Known Traffic Violations

Experiment Design. In this experiment, we evaluated the effectiveness and efficiency of VioHawk in reproducing the 42 known traffic violations collected in $Dataset_{GT}$ (see §5.1). Here, we tried to demonstrate the advantages of VioHawk by comparing it with six baseline tools (see §5.1). Since VioHawk and these baseline tools all take initial simulation scenarios (seeds) as inputs to activate the violation identification. Hence, we additionally prepared 42 initial simulation scenarios for them (i.e., one initial scenario provided for the reproduction of each traffic violation). Notably, these initial scenarios were not meticulously crafted. As showcased in Figure 7, they only contain the necessary road structures, the EGO vehicle and the least number of traffic participants that satisfy the law-specified driving conditions. Given the initial scenarios, we used each tool to run 42 fuzzing tasks, and the time limit for each fuzzing task was set to 3 CPU hours. During the fuzzing testing, we considered two evaluation metrics: ❶ **Time to Exposure (TTE)** quantifies the time required to trigger the desired traffic violation; ❷ **# of Seed Queue** measures the number of generated scenarios before triggering the violation or reaching the time limit.

Experiment Results. As shown in Table 6, VioHawk successfully reproduced 40/42 traffic violations within the time limit. For comparison, baseline tools could reproduce at most 13/42 traffic violations. Furthermore, unfortunately, DriveFuzz [37] and AV-Fuzzer [42] failed to identify any of the ground-truth violations collected in $Dataset_{GT}$. For the violations successfully reproduced by both

VioHawk and baseline tools, VioHawk could save 1.6X~8.9X the reproduction time. On average, VioHawk only required about 15.0 rounds of mutations to trigger the desired traffic violation. These key results clearly demonstrate the high effectiveness of VioHawk, due to the proposed area-based criticality-guided fuzzing.

False Positives. To ensure fair comparisons with existing works, each baseline tool is enhanced with our violation oracles to observe the occurrence of target traffic violations (see §5.1). When analyzing the experiment results listed in Table 6, we manually confirmed that no false positives were reported. This is mainly because our violation oracles are implemented based on the area-based traffic law formalization, which strictly encodes the violation conditions.

As described below, we also conducted a thorough breakdown analysis on the fuzzing results of each tool.

Breakdown Analysis: VioHawk. Despite the high effectiveness in violation detection, VioHawk still failed to reproduce 2/42 traffic violations collected in $Dataset_{GT}$. After careful analysis, we confirmed that the 2 failures were caused by implementation issues rather than design flaws. To be specific, Commonroad-Reach [36], the third-party library that VioHawk integrates to calculate the drivable areas, currently cannot handle complex high-curvature four-lane roads. Also, when mutating the driving maneuvers of NPC vehicles at an intersection, LGSVL APIs (i.e., APIs provided by the simulation platform) cannot correctly set the target lane to follow. In the future, we would fix these implementation issues to further improve the usability of VioHawk.

We further conducted breakdown analysis on the time cost of VioHawk. We found that the average time cost of drivable area estimation per time frame is only 1.63 seconds, and only about 17% of the total fuzzing time is used to calculate the drivable areas. To sum up, our area-based formalization of traffic laws is quite computationally efficient, and meanwhile can offer high-quality guidance for scenario mutations. Besides, as introduced in §4.4, 3 of the 5 mutation operators utilized by VioHawk can only be applied at the initial time frame of the scenario, rather than the most critical frame. Here, we also conducted breakdown analysis to investigate the consequences of this design choice. Finally, we found that only 75/426 (i.e., 17.6%) T_0-applied mutation operators might unexpectedly cause a decrease in the criticality scores of the scenarios. Hence, we argue that the consequences of these ineffective mutations are quite tolerable and can be handled by our criticality-based fuzzing scheduling mechanism.

Breakdown Analysis: LawBreaker and ABLE. Among all baselines, LawBreaker [51] and ABLE [56] have the best performance in violation reproduction. Since ABLE [56] enhances LawBreaker [51] with optimized GFlowNet model [20] and active learning technique, it has comparably better performance than LawBreaker [51]. Generally, reasons are two-fold for the unsatisfactory effectiveness or efficiency of these two tools: ❶ **Incorrect quantification of scenario criticality that misleads fuzzing scheduling.** When producing the violation scenarios of **R-1** and **R-2** (i.e., traffic-light-related laws originally supported by these two baselines), LawBreaker [51] and ABLE [56] mainly consider the distance to the stop line and the EGO speed to quantify the scenario criticality. However, these discrete indicators cannot reflect the scenario criticality when the traffic light is green. In comparison, under the area-based formalization of traffic laws leveraged by VioHawk, the states of traffic lights are modeled as the necessary hazardous state for the traffic violation. ❷ **Blind Scenario Mutations.** In most cases, LawBreaker [51] and ABLE [56] cannot identify violation scenarios even with about 200 rounds of mutations. Through manual case studies, we found that they indeed could generate critical scenarios (i.e., quite close to the scenarios implying ADS traffic violations). However, without deterministic mutation guidance, it is quite hard for them to finally trigger the desired violations.

Breakdown Analysis: DriveFuzz, AV-Fuzzer, AutoFuzz and SAMOTA. Similar to LawBreaker [51] and ABLE [56], these baseline tools also lack deterministic mutation guidance, consequently causing unsatisfactory performance. Worse still, without the capabilities of the formalization of complex traffic laws, these tools can hardly navigate the scenario searching process towards the target violations.

5.3 Identifying Unknown Traffic Violations

Experiment Design. To further demonstrate the utility of VioHawk in traffic law compliance testing, we chose another ten traffic laws for evaluation (listed in Table 7). Among these laws, six of them are newly collected from the N.Y. Driver's Manual [11] and the Chinese traffic laws [14, 15], which are not considered in existing works [37, 51, 55]. Besides, the remaining four of them are traffic laws considered by existing works [37, 51, 55], but no violation scenarios have been identified. Similar to the experiment presented in §5.2, we accordingly prepare the basic initial scenarios as inputs of VioHawk, and the time limit for each fuzzing task is set to 3 hours. Notably, when violation scenarios were identified on Apollo 7.0, we also tried to reproduce this scenario on Apollo 8.0 (i.e., the latest version of Apollo), so as to investigate the evolution trends of ADS against traffic law compliance.

Experiment Results. As summarized in Table 7, with the help of VioHawk, we found that 9/10 traffic laws could be violated on Apollo 7.0, and 8/9 violation scenarios can be reproduced even on the latest version of Apollo (i.e., Apollo 8.0). The 1/9 reproduction failure is because Apollo 8.0 has deployed patches [13] to address this issue. However, the remaining 8/9 reproducible violations indicate that developers have not promptly addressed these traffic violations. In the following, we also performed code-level case studies to confirm the corresponding root causes.

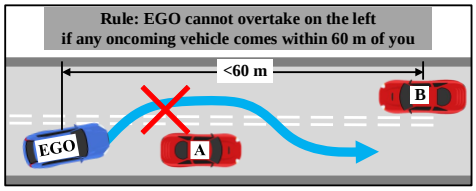
Case Study (NR-5): Incautious Overtaking. According to N.Y. Driver's Manual [11], when overtaking by borrowing the reverse lane, EGO should be careful of oncoming vehicles, and shall not overtake if the residual distance is less than 60 meters. However, as shown in Figure 8(B), when overtaking on a borrowed lane, Apollo only considers avoiding collision with the NPC in the borrowed lane, ignoring the distance limit between the oncoming NPC and EGO. This deficiency finally leads to an incautious overtaking behavior.

6 DISCUSSION

Usability of VioHawk. As shown in Table 5 (i.e., traffic laws collected in $Dataset_{GT}$), given a traffic law under test, we should first manually determine which pre-defined oracles (see Table 1,

Table 7: Previously Unknown Traffic Violations of Apollo.

Type	ID	Source	Description	Violation Detected by VioHawk	
				Apollo 7.0	Apollo 8.0
Right-Of-Way	NR-1	[55]&U.S. Law	Vehicles approaching the intersection must give way to vehicles arriving earlier;	✓	✓
	NR-2	[55]&U.S. Law	Vehicles should give way to pedestrians when turning at an intersection;		
	NR-3	U.S. Law	Vehicles must decelerate when approaching the crosswalk;	✓	✓
	NR-4	[51]&Chinese Law	Vehicles shall not enter a blocked intersection;	✓	✓
Overtaking	NR-5	U.S. Law	Vehicles shall not overtake if the oncoming vehicle is within 200 feet;	✓	✓
	NR-6	Chinese Law	Vehicles shall not overtake on the bend;	✓	✓
	NR-7	U.S. Law	Vehicles shall not overtake across the solid yellow line;	✓	
Parking	NR-8	U.S. Law	Vehicles shall not park near a parked vehicle;	✓	✓
	NR-9	[55]&U.S. Law	Vehicles shall not park in the intersection;	✓	✓
	NR-10	Chinese Law	Vehicles shall not park on the bend;	✓	✓



(A) Traffic Violation: Incautious Overtaking Decision

```
front_npc = get_front_npc()
# judge if the front NPC  is blocking the road
if front_npc and is_npc_blocking_road(front_npc):
        adjacent_npc = get_adjacent_npc()
        # judge if ego will collide with the adjacent npc
        if collision(pred_traj(adjacent_npc),planned_traj(ego)):
                # ego decelerates to avoid collision
                ego.decelerate(adjacent_npc)
```

(B) Pseudo Code of Root Cause in Apollo 7.0

Figure 8: Case Study on Violation Scenario of NR-5 (Table 7).

Table 2 and Table 3) should be used to achieve the area-based traffic law formalization. To further improve the usability of VioHawk, a possible solution is to automate this procedure by leveraging advanced AI-based techniques (e.g., Large Language Models).

Applicability of VioHawk to Different Laws. As clarified in §2.3, VioHawk is compatible with traffic laws that constrain diverse driving maneuvers (e.g., parking, braking, turning, overtaking, etc.) of vehicles. For instance, among the 50 real-world traffic laws considered by existing works [51, 55, 56], we confirmed that 43/50 traffic laws are related to driving maneuvers, and all of them can be modeled by VioHawk.

Completeness of VioHawk. Considering the complex nature of ADS functionalities and the inexhaustible search space of driving scenarios, it remains an open challenge to achieve complete verification of ADS safety. Generally, VioHawk is a sound while incomplete bug-hunting tool for ADS testing rather than a complete verification tool. However, we believe that VioHawk can still drive practical value for the community. Our case study (see §5.3) demonstrates that the violations reported by VioHawk can explicitly facilitate the diagnosis and mitigation of ADS faults.

Lesson Learned. In evaluation, we observed that Apollo only considers some basic traffic rules (e.g., traffic-light-related) to ensure traffic law compliance. Code implementations for the compliance of other traffic laws are either omitted or scattered across different source files. These improper implementation practices would make it difficult for ADS developers to debug and fix safety-violation issues, leaving various bugs unpatched across versions (see §5.3). To tackle this issue, we strongly suggest that a standardized and independent module can be developed and deployed in ADS for guarding traffic law compliance, which can not only improve the maintainability of ADS code, but also ensure the transferability of ADS across regions with different traffic laws.

7 RELATED WORK

Scenario-based ADS testing. Most existing works in ADS simulation testing are devoted to detecting general ADS accidents (e.g., collisions) [23, 27, 28, 30, 33, 40, 41, 43, 48, 52, 53, 57]. While [31, 32, 37, 51, 56] have ventured into the detection of traffic law violations, as detailed in §2.3, they cannot effectively identify violation scenarios without deterministic guidance on scenario mutations. VioHawk, however, can meticulously induce the EGO vehicle to drive into the law-specified hazardous areas (i.e., to violate a law).

Code-level ADS Verification. Formal verification methods [18] leverage mathematical techniques (e.g., mathematical formulas) to define and assess complex specifications for verifying the compliance of hardware and software systems [25, 26, 35, 38, 39]. Recently, AVChecker [55] has applied them to verify traffic law compliance of ADS code. However, without giving simulation scenarios of violation as proof, it inevitably reports false alarms.

8 CONCLUSION

In this work, we introduce VioHawk, a simulation-based ADS fuzzer designed for detecting traffic violations of high-level ADSs. VioHawk works by formalizing traffic law regulations as hazardous driving areas on the map, and applying mutation operations on the scenario configurations to induce the autonomous vehicle to drive into law-specified hazardous areas (i.e., to violate a law). We conducted extensive experiments to demonstrate the advantages of VioHawk compared to existing tools in identifying traffic violations of Apollo (i.e., one of the most advanced high-level ADSs). With the help of VioHawk, we identified 9+8 previously unknown violations against real-world traffic laws on Apollo 7.0/8.0.

9 DATA AVAILABILITY

Our prototype, implementations of baseline tools, benchmark dataset and identified ADS violations are all available at https://anonymous.4open.science/r/VioHawk-ISSTA24-75DA.

REFERENCES

[1] LGSVL Python API. https://www.svlsimulator.com/docs/#python-api.
[2] American Fuzzy Lop. http://lcamtuf.coredump.cx/afl, 2023.
[3] Apollo: An Open Autonomous Driving Platform. https://github.com/ApolloAuto/apollo, 2023.
[4] Autonomous Vehicle Collision Reports. https://www.dmv.ca.gov/portal/vehicle-industry-services/autonomous-vehicles/autonomous-vehicle-collision-reports/, 2023.
[5] Autoware-AI. https://github.com/autowarefoundation/autoware, 2023.
[6] BeamNG.tech. https://beamng.tech/services/#adas-autonomous-driving/, 2023.
[7] Companies That Develop Autonomous Driving. https://aimagazine.com/technology/top-10-companies-developing-autonomous-vehicle-technology, 2023.
[8] ISO 21448. https://www.iso.org/standard/77490.html, 2023.
[9] ISO 34502:Road vehicles–Test scenarios for Automated Driving Systems–Scenario Based Safety Evaluation Framework. https://www.iso.org/standard/78951.html, 2023.
[10] LGSVL 2021.2.2. https://github.com/lgsvl/simulator/tree/release-2021.2, 2023.
[11] New York State Driver's Manual & practice tests. https://dmv.ny.gov/driver-license/drivers-manual-practice-tests, 2023.
[12] OSMIUM:OpenStreetMap Parser. https://github.com/osmcode/pyosmium, 2023.
[13] Patch for the Double-Yellow-Line Check. https://github.com/ApolloAuto/apollo/pull/13629, 2023.
[14] Regulations for the Implementation of the Road Traffic Safety Law of the People's Republic of China. https://www.gov.cn/gongbao/content/2004/content_62772.htm, 2023.
[15] Road Traffic Safety Law of the People's Republic of China. https://www.gov.cn/banshi/2005-08/23/content_25575.htm, 2023.
[16] SanFrancisco Map in LGSVL. https://github.com/lgsvl/SanFrancisco, 2023.
[17] Shapely. https://github.com/shapely/shapely, 2023.
[18] Wiki: Formal Methods. https://en.wikipedia.org/wiki/Formal_methods, 2023.
[19] Wiki: OpenStreetMap. https://en.wikipedia.org/wiki/OpenStreetMap, 2023.
[20] E. Bengio, M. Jain, and et al. Flow Network based Generative Models for Non-iterative Diverse Candidate Generation. *NeurIPS*, 2021.
[21] Y. Cao, S. H. Bhupathiraju, and et al. You Can't See Me: Physical Removal Attacks on LiDAR-based Autonomous Vehicles Driving Frameworks. In *USENIX Security*, 2023.
[22] Y. Cao, N. Wang, and et al. Invisible for Both Camera and Lidar: Security of Multi-sensor Fusion Based Perception in Autonomous Driving under Physical-world Attacks. In *SP*, 2021.
[23] M. Cheng, Y. Zhou, and et al. BehAVExplor: Behavior Diversity Guided Testing for Autonomous Driving Systems. In *ISSTA*, 2023.
[24] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An Open Urban Driving Simulator. In *Conference on Robot Learning*, 2017.
[25] G. E. Fainekos. Revising Temporal Logic Specifications for Motion Planning. In *ICRA*, 2011.
[26] G. E. Fainekos, H. Kress-Gazit, and et al. Temporal Logic Motion Planning for Mobile Robots. In *ICRA*, 2005.
[27] A. Gambi, T. Huynh, and G. Fraser. 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*, pages 257–267, 2019.
[28] A. Gambi, M. Mueller, and G. Fraser. Automatically testing self-driving cars with search-based procedural content generation. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 318–328, 2019.
[29] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., 1989.
[30] J. C. Han and Z. Q. Zhou. Metamorphic Fuzz Testing of Autonomous Vehicles. In *ICSE Workshops*, 2020.
[31] F. U. Haq, D. Shin, and L. 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.
[32] F. U. Haq, D. Shin, and L. C. Briand. Many-objective reinforcement learning for online testing of dnn-enabled systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1814–1826. IEEE, 2023.
[33] Z. Hu, S. Guo, and et al. Coverage-based Scene Fuzzing for Virtual Autonomous Driving Testing. *arXiv preprint arXiv:2106.00873*, 2021.
[34] Y. Huai, Y. Chen, and et al. Doppelgänger Test Generation for Revealing Bugs in Autonomous Driving Software. In *ICSE*, 2023.
[35] K. Hyungsub, O. M. Ozgur, and et al. PGFUZZ: Policy-guided Fuzzing for Robotic Vehicles. In *NDSS*, 2021.
[36] E. Irani Liu, G. Würsching, and et al. CommonRoad-Reach: A Toolbox for Reachability Analysis of Automated Vehicles. In *ITSC*, 2022.
[37] S. Kim, M. Liu, and et al. DriveFuzz: Discovering Autonomous Driving Bugs through Driving Quality-Guided Fuzzing. In *CCS*, 2022.
[38] H. Kress-Gazit, G. E. Fainekos, and et al. Where's waldo? Sensor-based Temporal Logic Motion Planning. In *ICRA*, 2007.
[39] M. Lahijanian, J. Wasniewski, and et al. Motion Planning and Control From Temporal Logic Specifications with Probabilistic Satisfaction guarantees. In *ICRA*, 2010.
[40] C. Li, C.-H. Cheng, and et al. ComOpT: Combination and Optimization for Testing Autonomous Driving Systems. In *ICRA*, 2022.
[41] C. Li, J. Sifakis, and et al. Simulation-Based Validation for Autonomous Driving Systems. *ISSTA*, 2023.
[42] G. Li, Y. Li, and et al. AV-FUZZER: Finding Safety Violations in Autonomous Driving Systems. In *ISSRE*, 2020.
[43] C. Lu, Y. Shi, and et al. Learning Configurations of Operating Environment of Autonomous Vehicles to Maximize Their Collisions. *TSE*, 2022.
[44] Z. Peng, J. Yang, and et al. A First Look at the Integration of Machine Learning Models in Complex Autonomous Driving Systems: A Case Study on Apollo. In *FSE*, 2020.
[45] V. Riccio and P. Tonella. Model-based exploration of the frontier of behaviours for deep learning system testing. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 876–888, 2020.
[46] G. Rong, B. H. Shin, and et al. Lgsvl Simulator: A High Fidelity Simulator for Autonomous Driving. In *ITSC*, 2020.
[47] J. Shao. Testing Object Detection for Autonomous Driving Systems via 3D Reconstruction. In *ICSE-Companion*, 2021.
[48] R. Song, M. O. Ozmen, and et al. Discovering Adversarial Driving Maneuvers Against Autonomous Vehicles. In *USENIX Security*, 2023.
[49] S. Söntges and M. Althoff. Computing Possible Driving Corridors for Automated Vehicles. In *IV*, 2017.
[50] J. Sun, Y. Cao, and et al. Towards Robust Lidar-based Perception in Autonomous Driving: General Black-box Adversarial Sensor Attack and Countermeasures. In *USENIX Security*, 2020.
[51] Y. Sun, C. M. Poskitt, and et al. LawBreaker: An Approach for Specifying Traffic Laws and Fuzzing Autonomous Vehicles. In *ICSE*, 2022.
[52] H. Tian, Y. Jiang, and et al. MOSAT: Finding Safety Violations of Autonomous Driving Systems Using Multi-objective Genetic Algorithm. In *ESEC/FSE*, 2022.
[53] S. Wang, Z. Sheng, and et al. ADEPT: A Testing Platform for Simulated Autonomous Driving. In *ASE*, 2022.
[54] C. Yan, Z. Xu, and et al. Rolling Colors: Adversarial Laser Exploits Against Traffic Light Recognition. In *USENIX Security*, 2022.
[55] Q. Zhang, D. K. Hong, and et al. A Systematic Framework to Identify Violations of Scenario-dependent Driving Rules in Autonomous Vehicle Software. *ACM on Measurement and Analysis of Computing Systems*, 2021.
[56] X. Zhang, W. Zhao, Y. Sun, J. Sun, Y. Shen, X. Dong, and Z. Yang. Testing automated driving systems by breaking many laws efficiently. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 942–953, 2023.
[57] Z. Zhong, G. Kaiser, and et al. Neural Network Guided Evolutionary Fuzzing for Finding Traffic Violations of Autonomous Vehicles. *TSE*, 2022.
[58] Y. Zhou, Y. Sun, and et al. Specification-based Autonomous Driving System Testing. *TSE*, 2023.
[59] T. Zohdinasab, V. Riccio, A. Gambi, and P. Tonella. Deephyperion: exploring the feature space of deep learning-based systems through illumination search. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 79–90, 2021.