

Singapore Management University

## Institutional Knowledge at Singapore Management University

---

Research Collection School Of Computing and Information Systems

School of Computing and Information Systems

---

7-2023

### Testing automated driving systems by breaking many laws efficiently

Xiaodong ZHANG  
*Xidian University*


Wei ZHAO  
*Singapore Management University, weizhao.2021@phdcs.smu.edu.sg*


Yang SUN  
*Singapore Management University, yang.sun.2023@mi.smu.edu.sg*

Jun SUN  
*Singapore Management University, junsun@smu.edu.sg*

Yulong SHEN  
*Xidian University*

Follow this and additional works at: [https://ink.library.smu.edu.sg/sis\\_research](https://ink.library.smu.edu.sg/sis_research)

 See next page for additional authors

 Part of the [Software Engineering Commons](#), [Transportation Commons](#), and the [Transportation Law Commons](#)

---

#### Citation

ZHANG, Xiaodong; ZHAO, Wei; SUN, Yang; SUN, Jun; SHEN, Yulong; DONG, Xuewen; and YANG, Zijiang. Testing automated driving systems by breaking many laws efficiently. (2023). *ISSTA 2023: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, Seattle, WA, July 17-21*. 942-953.

Available at: [https://ink.library.smu.edu.sg/sis\\_research/8078](https://ink.library.smu.edu.sg/sis_research/8078)

This Conference Proceeding Article is brought to you for free and open access by the School of Computing and Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Computing and Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email [cherylids@smu.edu.sg](mailto:cherylids@smu.edu.sg).

---

**Author**

Xiaodong ZHANG, Wei ZHAO, Yang SUN, Jun SUN, Yulong SHEN, Xuewen DONG, and Zijiang YANG



# Testing Automated Driving Systems by Breaking Many Laws Efficiently

Xiaodong Zhang  
zhangxiaodong@xidian.edu.cn  
Xidian University  
Xi'an, China  
Singapore Management University  
Singapore

Wei Zhao  
weizhao.2021@phdcs.smu.edu.sg  
Singapore Management University  
Singapore

Yang Sun  
yangsun.2020@phdcs.smu.edu.sg  
Singapore Management University  
Singapore

Jun Sun\*  
junsun@smu.edu.sg  
Singapore Management University  
Singapore

Yulong Shen  
ylshen@mail.xidian.edu.cn  
Xidian University  
Xi'an, China

Xuwen Dong  
xwdong@xidian.edu.cn  
Xidian University  
Xi'an, China

Zijiang Yang  
yang@guardstrike.com  
GuardStrike Inc.  
Xi'an, China

## ABSTRACT

An automated driving system (ADS), as the brain of an autonomous vehicle (AV), should be tested thoroughly ahead of deployment. ADS must satisfy a complex set of rules to ensure road safety, e.g., the existing traffic laws and possibly future laws that are dedicated to AVs. To comprehensively test an ADS, we would like to systematically discover diverse scenarios in which certain traffic law is violated. The challenge is that (1) there are many traffic laws (e.g., 13 testable articles in Chinese traffic laws and 16 testable articles in Singapore traffic laws, with 81 and 43 violation situations respectively); and (2) many of traffic laws are only relevant in complicated specific scenarios.

Existing approaches to testing ADS either focus on simple oracles such as no-collision or have limited capacity in generating diverse law-violating scenarios. In this work, we propose ABLE, a new ADS testing method inspired by the success of GFlowNet, which Aims to Break many Laws Efficiently by generating diverse scenarios. Different from vanilla GFlowNet, ABLE drives the testing process with dynamically updated testing objectives (based on a robustness semantics of signal temporal logic) as well as active learning, so as to effectively explore the vast search space. We evaluate ABLE based on Apollo and LGSVL, and the results show that ABLE outperforms the state-of-the-art by violating 17% and 25% more laws when

testing Apollo 6.0 and Apollo 7.0, most of which are hard-to-violate laws, respectively.

## CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging.**

## KEYWORDS

Generative Flow Network, Traffic Laws, Automated Driving System, Baidu Apollo, Testing Scenario Generation

## ACM Reference Format:

Xiaodong Zhang, Wei Zhao, Yang Sun, Jun Sun, Yulong Shen, Xuwen Dong, and Zijiang Yang. 2023. Testing Automated Driving Systems by Breaking Many Laws Efficiently. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '23)*, July 17–21, 2023, Seattle, WA, USA. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3597926.3598108>

## 1 INTRODUCTION

The rapid development of autonomous driving technology is exciting as it promises tremendous social and economic benefits by reducing traffic accidents and improving traffic efficiency [19, 49]. Ensuring safety is one of the primary considerations before the wide-scale deployment of AVs. An investigation from US National Highway Traffic Safety Administration concludes that AVs only can prevent up to 34% of traffic crashes and even introduces accidents if the automated driving technology does not eliminate the traffic violations [38]. For example, when a Tesla car with Full Self-Driving feature is set as “assertive” operating mode, it allows vehicles to illegally roll through stop signs at 4-way intersection at speeds of up to 5.6 MPH. As the illegal rolling will greatly increase the risk of a crash, Tesla has to recall all the cars with this feature [13]. An ADS must strictly obey traffic laws during self-driving, as traffic laws are the current standard formula for road safety [16] (at least

\*This is the corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

ISSTA '23, July 17–21, 2023, Seattle, WA, USA

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0221-1/23/07.

<https://doi.org/10.1145/3597926.3598108>

until traffic laws dedicated to AVs are developed). However, it is challenging to manually design testing scenarios to test an ADS against intricate traffic laws. The challenge is that (1) there are many traffic laws (e.g., 13 testable articles in Chinese traffic laws and 16 testable articles in Singapore traffic laws, with 81 and 43 violation situations respectively [4]); and (2) many of traffic laws are only relevant in complicated scenarios.

Recently, researchers have proposed multiple approaches to efficiently generate scenarios for testing ADS [18, 23, 29, 32, 34]. These approaches leverage random sampling or randomized searching to generate scenarios, such as genetic algorithm (GA) [34] and machine learning [32]. Unfortunately, these techniques often adopt a fairly weak oracle, i.e., no-collision or reaching destination. We foresee that human drivers and AVs will share the road system in the near future. It is thus expected that AVs would follow the existing traffic laws. This is not only because these laws are designed for road safety but also because human drivers drive with the expectation that other vehicles follow the same laws. To violate traffic laws comprehensively is not easy as multiple factors are often required to be satisfied simultaneously. Consider the third case of Article #38 in the *Regulations for the Implementation of the Road Traffic Safety Law of the People's Republic of China* [21]. It describes that when the red light is on, vehicles are prohibited from passing. Except that, vehicles turning right can pass without hindering the passage of vehicles or pedestrians. Obviously, existing approaches focus on testing for collision is unlikely helpful in generating scenarios for violating such laws.

Only a few works focus on testing ADS against traffic laws in the literature. The state-of-the-art is the recent work LawBreaker [48], which adopts a GA-based testing method. Our investigation shows that LawBreaker finds almost all the failures during the initial 50 tests, e.g., only 10% of the failures are discovered with the remaining 462 tests, which costs 9 hours of simulation. The reason is that LawBreaker tends to repeatedly discover scenarios that are similar to those already-found ones as the GA approach shows premature convergence [41]. This is in terms due to the vast search space of testing scenarios and the fact that different law-violating scenarios are often far away from each other (i.e., requiring very different scenarios). While some works, such as SAMOTA [23], also claim testing ADS against traffic laws, their approaches are very restrictive, i.e., only testing compliance with signal lights. To better test an ADS against various traffic laws, we must improve the diversity of the generated scenarios, i.e., distinct ways of violating traffic laws. To avoid generating redundant testing scenarios (each of which is costly to simulate with existing simulators such as LGSVL [45]), we should generate diverse scenarios violating different laws as efficiently as possible. Furthermore, the identified law-violating scenarios must be tested physically eventually, which is very costly.

In this work, we propose an approach called ABLE to generate diverse scenarios for testing ADS against real-world traffic laws. ABLE models scenario generation as sampling a sequence of constructive actions (i.e., parameter assignment), based on a recently proposed model called GFlowNet [11]. Note that GFlowNet was designed for a different setting (i.e., generating promising protein sequences [26]) and we thus have to enhance GFlowNet in multiple ways, e.g., with dynamically updating objective functions which are computed based on a robustness semantics of signal temporal

logic. Furthermore, in order to generate valid scenarios (which are subject to physical laws as well as environmental constraints), we must constrain the search using appropriate types and invariants.

We have implemented ABLE and evaluated it using a scenario scripting language called AVUnit [6], LGSVL as the simulator, and multiple versions of Baidu Apollo [9, 10] as the ADSs under test. ABLE finds a total of 117 and 130 different violations of 13 Chinese traffic laws when testing Apollo 6.0 and Apollo 7.0, respectively. Compared to LawBreaker, ABLE outperforms the state-of-the-art by violating 17% and 25% more laws. A close investigation shows that these newly discovered violations are missed by LawBreaker because they require complicated scenarios and are hard to discover through random searches. We further conduct experiments to show that our improvements to the vanilla GFlowNet model are relevant and effective. The key contributions of this work are as follows.

- We propose a new scenario generation method, to generate diverse testing scenarios which trigger violation of real-world traffic laws.
- To better adapt to scenario generation, we have made three optimizations to GFlowNet. The first is that we define rewards based on the robustness degree of a scenario with respect to the specification; The second is to support dynamic rewards to keep targeting the uncovered violations during active learning; the third is to incorporate domain knowledge so that only appropriate actions at each sample step are selected.
- We have implemented ABLE based on Apollo and LGSVL, and conducted multiple comparative experiments against the state-of-the-art.

This paper is organized as follows: Section 2 provides an illustrative example to show how ABLE works. Section 3 defines the problem. Section 4 presents the detailed algorithm. Section 5 evaluates ABLE with multiple experiments. Section 6 presents the threats to validity. Lastly, Section 7 reviews related works, which is then followed by a conclusion in Section 8.

## 2 ILLUSTRATIVE EXAMPLE

List 1 presents an example traffic law specification written in signal temporal logic (STL)[24, 43]. In particular, it describes the 7-th clause in Article #51 of the *Regulations for the Implementation of the Road Traffic Safety Law of the People's Republic of China*[21], which stipulates how a vehicle should behave at an intersection without a directional signal. An English translation of Article #51-7 reads as follows: *At intersections with no direction indicator lights, turning motor vehicles must give priority to non-turning vehicles and pedestrians. Right-turning motor vehicles traveling in the opposite direction must give priority to left-turning vehicles.*

Article #51-7 is specified as `law51_sub7` in List 1. In particular, clause `law51_sub7_1` specifies that the ego vehicle may be either turning right or left. Clause `law51_sub7_2` specifies that there is an NPC (Non-Player Character) vehicle or pedestrian with priority moving in the relevant lane. Clause `law51_sub7_3` specifies that the ego vehicle must slow down to less than 0.5m/s within the next  $d$  time steps. Note that F and G mean ‘eventually’ and ‘always’ respectively, which are modal operators in temporal logic. The

```

1 law51_sub7_1 = (direction==Right | direction==Left)
2 law51_sub7_2 = (PriorityNPCAhead==True |
   PriorityPedsAhead==True)
3 law51_sub7_3 = F[0,d](speed<0.5)
4 law51_sub7   = G((law51_sub7_1 & law51_sub7_2) ->
   law51_sub7_3))

```

**Listing 1: Formal specification of Article #51-7 in Chinese traffic laws**

```

1  $\bar{\theta}_{46}$ : F(((direction==Right)and(PriorityNPCAhead==True))
   and(G[0,d](not(speed<0.5))))
2  $\bar{\theta}_{47}$ : F(((direction==Right)and(PriorityPedsAhead==True))
   and(G[0,d](not(speed<0.5))))
3  $\bar{\theta}_{48}$ : F(((direction==Left)and(PriorityNPCAhead==True))and
   (G[0,d](not(speed<0.5))))
4  $\bar{\theta}_{49}$ : F(((direction==Left)and(PriorityPedsAhead==True))
   and(G[0,d](not(speed<0.5))))

```

**Listing 2: Four violation formulae of law51\_sub7 in List 1**

constant  $d$  can be customized by the user. In this example, we set  $d$  to 2 for simplicity.

It is challenging to check whether this traffic law law51\_sub7 will be violated by an ego vehicle. Its complicated formula involves multiple variables regarding different objects, and thus testing it requires multiple objects to be set up properly to reach a specific state at the same time. Furthermore, one article of the laws, due to its complexity, may be violated in many different ways, each of which should be tested adequately. For instance, List 2 shows four formulae<sup>1</sup>, which specify the four different ways of violating law51\_sub7. For example,  $\bar{\theta}_{46}$  captures a scenario where the ego vehicle does not give way during turning right when an NPC vehicle has priority on the road.

Most of the existing ADS testing methods[28, 32, 34, 37] cannot address the above problem, due to their inability to specify traffic laws formally. Although LawBreaker [48] is theoretically capable of testing the safety of ADS against traffic laws, according to our experiment, it fails to generate a scenario that demonstrates a violation of any of the four ways after 9 hours. In comparison, ABLE generates two scenarios which show two different ways of violating law51\_sub7, i.e.,  $\bar{\theta}_{46}$  and  $\bar{\theta}_{48}$ , in 3 and 5 hours respectively. By replaying the generated scenarios, it can be found that the ego vehicle does not slow down its speed at all when changing lane, whilst an NPC with priority is moving. Although there is no collision in the test, in reality many traffic accidents result from this unsafe lane or direction change, which is in the *Top Causes of Car Accidents*[42].

### 3 PROBLEM DEFINITION

An ADS must be tested with a wide range of scenarios before they can be deployed. Compared to real-world testing, simulation-based testing is more cost-effective and is more flexible [1]. For instance, simulator-based testing allows us to experiment with dangerous scenarios which are often life-threatening in real tests. It often can be done in the development phase on high-fidelity simulators, such as CARLA[15] and LGSVL[45]. Furthermore, the developers can design various scenario scripts to test whether an ADS obeys the safety specifications, such as traffic laws and no-collision. This

<sup>1</sup>Numbers 46-49 are the indexes of the four formulae in the total 81 violation formulae in [5].

**Table 1: Action types for each type of object**

Object Type	Action Types
Time	<b>set</b> {hour, minute}
Weather	<b>set</b> {rain,sunny,wetness,fog,snow,cloud}
Ego Vehicle	<b>set</b> {start position, start speed, destination position, destination speed }
NPC Vehicle	<b>set</b> {start position, start speed, waypoint position, waypoint speed, destination position, destination speed, vehicle type}
Pedestrian	<b>set</b> {start position, start speed, waypoint position, waypoint speed, destination position, destination speed }
Obstacle	<b>set</b> {position, size}

work thus focuses on the simulation testing of AV against traffic laws.

#### 3.1 Definition of Search Space

A test case of an ADS is typically referred to as a scenario, which intuitively specifies how different objects relevant to ADS testing evolve through time. There are mainly six types of objects in a scenario, i.e., ego vehicle, time, weather, NPC vehicles, pedestrians, and obstacles. Table 1 lists the allowed action types for the corresponding objects, each of which may affect the behavior of the ego vehicle. A scenario is a tuple  $\langle t, w, e, n, p, o \rangle$ , where each component is a sequence of actions for the corresponding object. The actions depend on the corresponding object participating in a traffic scenario.

Specifically,  $t$  is an action sequence on the time, e.g., set hour or set minutes.  $w$  is an action sequence on the weather, e.g., set rain or set fog.  $e$  is an action sequence on ego vehicle, e.g., set start point or set destination point.  $n$  is an action sequence on an NPC vehicle, e.g., set start speed or waypoint position.  $p$  is an action sequence, which has a similar setting as the NPC vehicle.  $o$  is an action sequence about the obstacle, e.g., set size or set position. Note that there are in general multiple NPC vehicles, pedestrians and obstacles in a scenario.

List 3 shows a scenario script written in AVUnit[6]. This script effectively defines a scenario where  $n$  consists of five NPC vehicles. Specifically, Line 1-3 specify the map, time, and weather in the simulation environment. In lines 5-8, the task of the ego vehicle is to move from the start point on lane\_540 to the target point on lane\_572, which are 50 and 60 meters away from their initial lane positions, respectively. Line 9-13 show that the Sedan car npc1 moves from the start position with an initial speed of 6 m/s to the target point. In particular, Line 11 adds a waypoint for npc1, which specifies its speed to be 4m/s when it arrives at 200 meters away from lane\_574 start point. The Other four NPC vehicles have a similar setting as npc1, which are omitted. All of these statements constitute a testing scenario. Note that a scenario can be mutated in many different ways. For instance, the value for setting the hour at can be 0, 1, ..., or 23, which means that 24 actions can be selected here. The offset of the start point for the ego vehicle can be set to any value within a predefined range, such as [40m, 60m]. The type of NPC vehicle can be set to any value in the set {BoxTruck,



```

1 map_name      = "san_francisco";
2 time         = 12:00;
3 weather      = {rain:0.5, fog:0.1, wetness:0.6};
4 evn         = Environment(time, weather);
5 ego_init_state = ("lane_540"->50);
6 ego_target_state = ("lane_572"->60);
7 vehicle_type = (car_model);
8 ego_vehicle  = AV(ego_init_state, ego_target_state,
                   vehicle_type);
9 npc1_type    = ("Sedan");
10 npc1_init_state = ("lane_574"->100, 6);
11 npc1_waypoint = ("lane_574"->200, 4);
12 npc1_dest_state = ("lane_569"->30);
13 npc1        = Vehicle(npc1_init_state,
                       npc1_waypoint, npc1_dest_state,
                       npc1_type);
14 (omit npc2-npc5)
15 npc_list    = {npc1, npc2, npc3, npc4, npc5};
16 scenario0   = CreateScenario{load(map_name),
                               ego_vehicle, npc_list, evn};

```

Listing 3: Code snippet of a scenario script

Hatchback, Jeep, SUV, SchoolBus, Sedan}, which are supported in a simulator.

### 3.2 Problem Definition

The formal specification of traffic laws is denoted as the symbol  $\Theta$ . We write  $\phi(\Theta)$  to denote the set of formulae that specify different ways in which  $\Theta$  can be violated, as defined in [48]. We refer the readers to [48] for details. Our testing objective is to generate one testing scenario at least for each of the violation formulae in  $\phi(\Theta)$ , i.e., to show different ways of violating each traffic law. Consider the examples in List 1 and 2 again. law51\_sub7 and violation formulae  $\{\theta_{46}, \theta_{47}, \theta_{48}, \theta_{49}\}$  are the set  $\Theta$  and  $\phi(\Theta)$ , respectively.

Our problem is to generate a set of scenarios  $\mathbb{V}\mathbb{S}$  such that the scenarios in  $\mathbb{V}\mathbb{S}$  satisfies as many formulae in  $\phi(\Theta)$  as possible. Note that not all of the formulas in  $\phi(\Theta)$  are satisfiable. There may be multiple reasons why a formula in  $\phi(\Theta)$  is not satisfiable. First, ideally it may be that the ADS is designed in such a way that it never violates the law, e.g., always stop before a red light. Secondly, satisfying a violation formula may require certain specific maps, which may not be available. For instance, Article 40 in [21] that regulates the behavior of vehicles when facing traffic lights with directions can not be tested if the map does not support traffic lights with directions.

## 4 ALGORITHM OF ABLE

The architecture of ABLE is shown in Figure 1. Firstly, Data Preparation prepares the data for training a model for guiding the testing in the next step, including transforming scenarios into action sequences and computing the rewards of scenarios according to their trace data. Secondly, Scenario Generation trains a generative model with the initial scenarios, then samples a batch of diverse scenarios based on the trained model. While the model is trained according to the recently proposed GFlowNet, we propose three optimizations so that it caters to the characteristics of our ADS testing problem. **Optimization 1** is that we define rewards based on the robustness degree of a scenario with respect to the specification, which intuitively measures how close the scenario is from satisfying the specification; **Optimization 2** is to dynamically update reward to keep targeting the uncovered violations, and **Optimization 3** is to

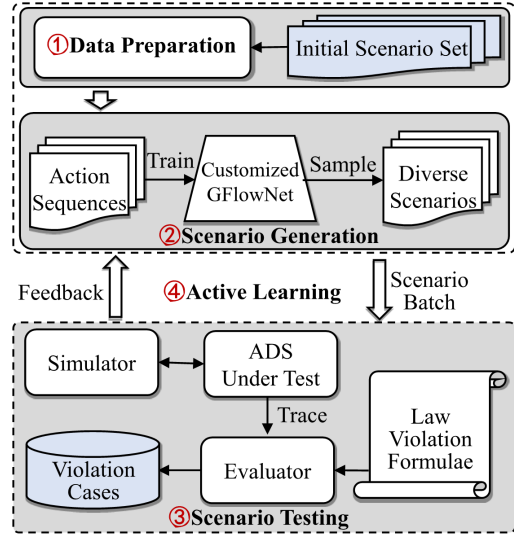


Figure 1: The architecture of ABLE

only choose the appropriate type of actions at each sample step to meet the characteristics of the scenario script. Thirdly, Scenario Testing iteratively runs the generated scenarios one-by-one in a simulator and then checks whether an ADS violates the traffic laws based on the traces generated from the scenarios. Lastly, Active Learning augments the scenario set with the newly-tested scenarios after each round and then updates the generative model.

### 4.1 Data Preparation

Algorithm 1 shows the details on how to process the data for the model training. It loads an initial set of scenarios (which are generated using existing approaches such as LawBreaker) and the set of formulae  $\phi(\Theta)$ , and outputs the processed action sequences associated with reward values. Each initial scenario is a scenario script written in a scenario description language such as AVUnit and a data trace obtained by simulating the scenario using a simulator such as LGSVL. For each scenario, the algorithm first encodes its scenario script as an action sequence as shown in Line 3, and then calculates its reward value over its trace based on  $\phi(\Theta)$  as shown in Line 4-8. The details are discussed below.

**4.1.1 Encoding Scenario Script.** A scenario script describes the scenario, i.e., associating values to different objects, such as time, weather, pedestrians, and NPC vehicles, as well as their concrete behaviors in terms of waypoints. The encoding should meet four goals. First, it must keep the context-sensitive nature of scenario scripts. Second, the encoded action sequence can be decoded to obtain the original scenario. Third, for the same operations to the same object in different scenario scripts, the converted actions must be consistent. Fourth, it must cover all the settings that are relevant to the testing results.

To meet the first and second goals, DataPreparation converts all the operations in a script into an action sequence, on which each action is sorted by its corresponding script location. To meet the

**Algorithm 1:** DataPreparation( $\phi(\Theta)$ , S)

---

**Input:**  $\phi(\Theta)$ : Violation formulae  
**S:** Scenario set with trace data  
**Output:**  $S^a$ : Action sequences for model training

```

1  $S^a = \emptyset$ ;
2 foreach  $\langle \text{scenario } sn, \text{trace } \pi_{sn} \rangle \in S$  do
3   actionSequence = encode( $sn$ );
4   robustnessVector = [];
5   foreach  $\bar{\theta} \in \phi(\Theta)$  do
6     | robustnessVector.add( $\rho(\bar{\theta}, \pi_{sn})$ );
7   end
8   reward = computeReward(robustnessVector);
9    $S^a$ .add( $\langle \text{actionSequence}, \text{reward} \rangle$ );
10 end
11 return  $S^a$ ;
```

---

	Type	Value	
3	set ego.start.lane_540	to 50	(Action1),
4	set ego.destination.lane_572	to 60	(Action2),
5	set npc1.type	to Sedan	(Action3),
6	set npc1.start.lane_574	to 100	(Action4),
7	set npc1.start.speed	to 6	(Action5),
8	set npc1.motion.lane_574	to 200	(Action6),
9	set npc1.motion.speed	to 4	(Action7),
10	set npc1.destination.lane_569	to 30	(Action8)
11	...	...	...

**Listing 4: The action sequence for the ego vehicle and npc1 in List 3**

third and fourth goals, DataPreparation encodes the action type with a unique prefix and builds a unique action identification by joining the prefix with the value. Consider the example in List 3. List 4 shows the encoded actions for ego and npc1. For example, Action1 describes the setting at Line 5 in List 3, and Action2 for Line 6 in List 3. For the added way point to npc1, Action6 and Action7 stand for the settings of offset and speed respectively.

**4.1.2 Computing Reward.** The reward is computed based on the robustness degree of each violation formula over the current trace as shown in Line 4-6 of Algorithm 1. In this work, the robustness degree is computed based on the quantitative semantics [14, 36, 40, 44] of STL, which evaluates how far a given trace is from satisfying an STL formula. Quantitative semantics is a measure of system safety that can be used to quantify the performance of a system with respects to a specification.[44]. As shown in Line 6, this measure is defined as a robustness degree function  $\rho$ , whose result is a real number. Consider a simple STL formula  $\theta = F[0, 6](\text{speed} > 3.5\text{m/s})$ . If speed is 3m/s within the 6 time steps in trace  $\pi$ ,  $\rho(\theta, \pi)$  is equal to -0.5, which indicates that  $\theta$  fails over  $\pi$ . If speed is 4m/s,  $\rho(\theta, \pi)$  is equal to 0.5, which indicates that  $\theta$  holds over  $\pi$ .

In this work, we design the reward function such that it satisfies two requirements. First, if a scenario covers many violation formulae with high robustness degree, its reward should be high, i.e., the reward can reflect the overall quality of a scenario. Second, the reward should be dynamically updated during the learning and fuzzing, i.e., a violation formula that has been satisfied will not be considered in the next iteration.

**Algorithm 2:** ScenarioGeneration( $S^a$ )

---

**Input:** Action sequences for training model  $S^a$   
**Output:** Scenario batch  $B$

```

1 Fit proxy  $\mathbb{R}$  on dataset  $S^a$ ;
2 Train generative model  $GFN$  using proxy  $\mathbb{R}$  on dataset  $S^a$ ;
3 Sample action sequence batch  $B_{seq}$  from  $GFN$ ;
4  $B = \text{decode}(B_{seq})$ ;
5 return  $B$ 
```

---

Line 8 computes the reward value of  $\pi_{sn}$  according to our new reward function in Equation 1, which presents a definition of the reward value for scenario  $sn$ .  $Cov(\bar{\theta})$  decides whether  $\bar{\theta}$  will be considered based on its coverage. If  $\bar{\theta}$  is covered, it will not be considered any more and  $Cov(\bar{\theta})$  is set as 0. Otherwise,  $Cov(\bar{\theta})$  is set as 1.  $Dis(\bar{\theta})$  measures how far violation formula  $\bar{\theta}$  will be satisfied, and its value depends on the robustness degree  $\rho(\bar{\theta}, \pi_{sn})$ . If  $\bar{\theta}$  is satisfied by trace  $\pi_{sn}$ , that is  $\rho(\bar{\theta}, \pi_{sn}) \geq 0$ ,  $Dis(\bar{\theta})$  is set as 0; Otherwise,  $Dis(\bar{\theta})$  is equal to  $-\rho(\bar{\theta}, \pi_{sn})$ .

$$\mathcal{R}(sn) = \sum_{\bar{\theta} \in \phi(\Theta)} \frac{Cov(\bar{\theta})}{Dis(\bar{\theta}) + 1}$$

$$Dis(\bar{\theta}) = \begin{cases} -\rho(\bar{\theta}, \pi_{sn}), & \rho(\bar{\theta}, \pi_{sn}) < 0 \\ 0, & \rho(\bar{\theta}, \pi_{sn}) \geq 0 \end{cases} \quad (1)$$

$$Cov(\bar{\theta}) = \begin{cases} 0, & \bar{\theta} \text{ is covered} \\ 1, & \bar{\theta} \text{ is not covered} \end{cases}$$

Consider the violation formulae in List 2 as the testing target. Let the scenario in List 3 be  $sn1$  and its trace be  $\pi_{sn1}$ . Over  $\pi_{sn1}$ , the robustness degree  $\rho(\{\bar{\theta}_{46}, \bar{\theta}_{47}, \bar{\theta}_{48}, \bar{\theta}_{49}\}, \pi_{sn1})$  equals  $\{-1.0, -2.0, -1.0, -2.0\}$ . Then,  $Dis(\{\bar{\theta}_{46}, \bar{\theta}_{47}, \bar{\theta}_{48}, \bar{\theta}_{49}\})$  is calculated to be  $\{1.0, 2.0, 1.0, 2.0\}$ .  $Cov(\{\bar{\theta}_{46}, \bar{\theta}_{47}, \bar{\theta}_{48}, \bar{\theta}_{49}\})$  is set to  $\{1, 1, 1, 1\}$  since the four violation formulae have not been covered. After calculation, the reward value  $\mathcal{R}(sn1)$  is 1.67.

## 4.2 Scenario Generation

Algorithm 2 presents the details on how testing scenarios are generated. It takes action sequences with reward values as input and outputs a batch of new high-reward scenarios.

**4.2.1 Training Model.** In order to return the evaluated reward of the samples instantly during training, a proxy is trained in advance. As shown in Line 1 of Algorithm 2, the proxy model  $\mathbb{R}_i$  is trained on the current data-set. This proxy model is a multi-layer neural network. The labels are the corresponding reward values of the scenarios. Without such a proxy model, the reward values of the candidates during training can only be obtained from the simulation, which would make training extremely time-consuming.

Line 2 shows that  $GFN_i$  would be trained with proxy  $\mathbb{R}_i$  on dataset  $S_i^a$ . We first define  $\mathcal{T}$  as a directed acyclic graph  $(\mathcal{A}, \mathcal{S}, \mathcal{E})$  with the set of actions  $\mathcal{A}$ , the set of nodes  $\mathcal{S}$  and the set of edges  $\mathcal{E}$ . An edge  $s \xrightarrow{a} s' \in \mathcal{E}$  corresponds a transition from state  $s$  to state  $s'$ , triggered by action  $a \in \mathcal{A}$ . After applying each action, a partially constructed object is generated, which can be called a node or a state  $s \in \mathcal{S}$ . Then, a learning objective should be set up.

The idea is that all the state flows and edge flows on the  $\mathcal{T}$  must comply with the flow matching criterion[11], i.e., the amount of flow entering any state equals the amount of flow coming out of it. Note that the flow can be interpreted as the probability mass on a state or an edge. This is achieved by defining a loss function  $\mathcal{L}$  as shown in Equation 2, whose global minimum gives rise to the flow consistency condition.

$$\mathcal{L} = \left( \log \frac{\sum_{s' \in \text{Parent}(s)} F(s' \rightarrow s)}{\sum_{s'' \in \text{Child}(s)} F(s \rightarrow s'')} \right)^2 \quad (2)$$

where  $s' \rightarrow s$  and  $s \rightarrow s''$  stands for an in-flow and an out-flow for state  $s$  respectively.

**4.2.2 Generating Scenarios.** As shown in Line 3 of Algorithm 2, a batch of action sequences is sampled from the well-trained model. The model can also be viewed as a sampling function with a stochastic strategy  $\lambda(a_t | s_t)$ , which outputs  $a_t$  according to current state  $s_t$  and then leads to new state  $s_{t+1}$ . An action sequence is constructed by starting from an initially empty state  $s_0$  and applying actions sequentially, and all complete trajectories must end in a special final state  $s_f$ . The construction of a scenario object  $s_n$  can be defined as a trajectory of states  $\langle s_0 \xrightarrow{a_0} s_1 \dots \xrightarrow{a_n} s_n \rightarrow s_f \rangle$ .

As the statements in a scenario script must be contextually relevant to each other, the corresponding actions should keep the same ordering as the statements. Thus, action ordering is fixed with respect to its type in an action sequence. This is a big difference with applying GFlowNet to the molecule synthesis domain[26], where a building block can appear multiple times in a molecule. However, in the testing scenario generation domain, only one action of the same type can be allowed on an action sequence. For example, if two time-setting actions appear on a sequence simultaneously, such as setting time to be 11 am and 23 pm, one of them will be redundant.

To avoid the above problem, we incorporate a simple domain model in our approach so as to only choose the appropriate type of action at each sample step during action sequence construction. This can be done by setting the probability of sampling unmatched-type actions as 0 while adding an action to the current sequence. Thus, the sampling strategy changes from  $\lambda(a_t | s_t)$  to  $\lambda(a_t | s_t, \text{type}(a_t) = T_t)$ , where  $\text{type}(a_t)$  denotes the action type of  $a_t$  and  $T_t$  denotes the desired action type in the  $t$ -th step on current sequence. In this work, constraint  $\text{type}(a_t) = T_t$  at different time steps is set statically according to the domain model. To keep our domain model simple and efficient, we do not consider correlation between actions across different time steps. This domain model dramatically reduces the search space from  $(|A_1| + |A_2| + \dots + |A_N|)^N$  to  $|A_1| * |A_2| * \dots * |A_N|$ , where  $N$  denotes the length of steps in a scenario and  $|A_i|$  denotes the number of actions in the  $i$ -th step. Consider the action sequence in List 4. When adding an action to Line 9, only the actions with type 'npc1.destination.lane\_569' can be chosen.

Finally, the algorithm decodes the action sequences to synthesize the executable scenario scripts as shown in Line 4 of Algorithm 2, which is the inverse of the encoding procedure. This is necessary as the simulators cannot recognize the generated action sequences. The synthesized scenarios will be fed into the simulator to drive further testing.

---

### Algorithm 3: ScenariosTesting( $B, \Theta$ )

---

**Input:**  $B$ : Testing Scenarios sampled from GFN  
 $\phi(\Theta)$ : Violation formulae  
**Output:**  
 $\Pi$ : Traces for the tested scenarios in  $B$   
 $\mathbb{V}\mathbb{S}_i$ : Law-violating scenarios in current batch

```

1  $\Pi = \emptyset$ ;
2 foreach  $sn \in B$  do
3    $\pi_{sn} = \text{Steer ADS to run scenario } sn \text{ in simulator and log}$ 
   the runtime trace data;
4   foreach  $\bar{\theta} \in \phi(\Theta)$  do
5     if  $\pi_{sn} \models \bar{\theta}$  then
6        $\phi(\Theta).\text{remove}(\bar{\theta})$ ;
7        $\mathbb{V}\mathbb{S}_i.\text{add}(\langle sn, \bar{\theta} \rangle)$ ;
8     end
9   end
10   $\Pi.\text{add}(\langle sn, \pi_{sn} \rangle)$ ;
11 end
12 return  $\Pi, \mathbb{V}\mathbb{S}_i$ ;

```

---

## 4.3 Scenario Testing

Algorithm 3 shows the details of how to test the ADS with the scenario batch. It inputs the newly-generated scenarios and violation formulae, and outputs the trace data and the new violations. For each scenario, we run the ADS in a simulator and log its run-time data as shown in Line 3. After a scenario is executed completely, we check whether each  $\bar{\theta}$  is satisfied over the logged trace  $\pi_{sn}$ , as shown in Line 4-7.

**4.3.1 Logging Run-time Data.** We call the run-time data at each time step a scene, which is a snapshot of the world (i.e., the status of all vehicles, pedestrians, and so on). A complete simulation of a scenario often have hundreds of time steps. Not all of the information in a scene is necessary for checking the violation of traffic laws. The algorithm just extracts the required data from the scene and assigns them to the corresponding variables, i.e., those constitute the violation formulae. Meanwhile, these variables also constitute the building blocks of our specification languages, i.e., the propositions.

Referring to work [48], we set up the following kinds of variables to log data: *Car Status Variables* describe the properties involving the lights, engine, horn, and direction of the AV, e.g., the variable *direction* in List 1; *Driving Status Variables* describe the speed, acceleration, and braking status of the AV, e.g., the variable *speed* in List 1; *Road Variables* capture properties of road that the AV is currently driving on, e.g., whether or not honking is allowed, the street light is on; *Traffic Signal Variables* allow for the specification of laws involving traffic lights at the junction an AV is approaching; *Traffic Variables* are associated with other vehicles sharing the road with the AV, as well as any pedestrians crossing it, e.g., the variables *PriorityNPCAhead* and *PriorityPedsAhead* in List 1; and *Map Variables* specify traffic laws related to environment conditions, e.g., the weather or time of day.



**4.3.2 Verifying Violation Formulae.** An execution trace  $\pi_{sn}$  is a sequence of scenes generated according to a scenario  $sn$ , denoted as  $\pi_{sn} = \{\alpha_0, \alpha_1, \dots, \alpha_n\}$ . A scene  $\alpha$  is a tuple of the form  $\alpha = (f_0, f_1, \dots, f_m)$ , where  $f$  is the valuation mapping of vehicles and surrounding traffic signal variables mentioned above. Considering  $\pi_{sn}$ , we write  $\pi_{sn} \models \bar{\theta}$  (resp.  $\pi_{sn} \not\models \bar{\theta}$ ) to denote that  $\bar{\theta}$  evaluates to be true (resp. false) over trace  $\pi_{sn}$ . Here,  $\pi_{sn} \models \bar{\theta}$  is measured with the robustness degree function  $\rho(\bar{\theta}, \pi_{sn})$ . If  $\rho(\bar{\theta}, \pi_{sn}) \geq 0$ , then  $\pi_{sn} \models \bar{\theta}$  evaluates to be true. Otherwise, it evaluates to be false.

If  $\pi_{sn} \models \bar{\theta}$  is satisfied, it indicates that ABLE finds a way to violate the traffic laws. Then,  $\bar{\theta}$  will be deleted from set  $\phi(\Theta)$  as shown in Line 6 of Algorithm 3, such that it will not be considered again in the subsequent iterations of fuzzing. Meanwhile, the scenario and its corresponding violation formula are also recorded in  $\mathbb{VS}_i$  in Line 7. After checking all the violation formulae, scenario  $s$  and its current trace  $\pi_{sn}$  are put into  $\Pi$  in pairs, which will be used in the next loop of active learning. Consider the four violation formulae in List 2. We get a newly-generated scenario  $sn2$  and its trace  $\pi_{sn2}$ , over which  $\rho(\{\bar{\theta}_{46}, \bar{\theta}_{47}, \bar{\theta}_{48}, \bar{\theta}_{49}\}, \pi_{sn2})$  is calculated to be  $\{0.0, -1.0, -1.0, -1.0\}$ . It is thus clear that the first violation formula  $\bar{\theta}_{46}$  is satisfied since  $\rho(\bar{\theta}_{46}, \pi_{sn2}) \geq 0$ .

## 4.4 Active Learning

The loop in Algorithm 4 integrates the above three components into an active learning framework. The algorithm inputs an initial scenario set, a set of violation formulae, and the number of active learning rounds, and outputs law-violating scenarios  $\mathbb{VS}$ . This active learning framework conducts two new tasks. One is to add the currently tested scenarios like  $sn2$  into the test set in Line 7, and another is to update the reward so that those already covered violated formulae will no longer be considered. Note that the updating operation is done at component `ScenarioTesting` as shown in Line 5. The former task not only augments the original scenario set, but also brings new high-reward modes for the next round of scenario generation. The latter one recomputes the reward values according to Equation 1 for all scenarios in the current scenario set, such that ABLE can re-target the uncovered violation formulae. We can call this a dynamic reward, which is more flexible and suitable for our scenario generation. The algorithm terminates when it reaches the bound of active learning loops. With the dynamic reward, active learning can identify those cases where multiple objectives are covered and then tackle the remaining ones one by one.

Consider scenarios  $sn1$  and  $sn2$ . As the violation formula  $\bar{\theta}_{46}$  has been covered,  $Cov(\{\bar{\theta}_{46}, \bar{\theta}_{47}, \bar{\theta}_{48}, \bar{\theta}_{49}\})$  is set to  $\{0, 1, 1, 1\}$ . Therefore, the reward values of  $sn1$  and  $sn2$  are recalculated to 1.17 and 1.5, respectively.

## 5 EVALUATION

We have implemented ABLE [5] based on the LGSVL simulation framework and use it to evaluate two latest versions of Baidu Apollo, i.e., Apollo 6.0 (released on September 22, 2020) and Apollo 7.0 (released on December 28, 2021). We use the data bridge in LawBreaker to spawn scenarios for the simulator and generate a trace using the collected data. We take the specification of 13 testable Chinese traffic laws in [48] as our testing target and transform it into 81

---

### Algorithm 4: ActiveLearning( $S_0, \phi(\Theta)$ )

---

**Input:**  $S_0$ : Initial scenario set with oracle  
 $\phi(\Theta)$ : A set of violation formulae that represent different law-violating ways  
 $L$ : The loop bound of active learning  
**Output:** Violation Scenarios  $\mathbb{VS}$

```

1  $\mathbb{VS} = \emptyset$ ;
2 for  $i = 0$  to  $L-1$  do
3    $S_i^a = \text{DataPreparation}(\phi(\Theta), S_i)$ ;
4    $B = \text{ScenarioGeneration}(S_i^a)$ ;
5    $\Pi, \mathbb{VS}_i = \text{ScenariosTesting}(B, \phi(\Theta))$ ;
6    $\mathbb{VS} = \mathbb{VS} \cup \mathbb{VS}_i$ ;
7    $S_{i+1} = S_i \cup \Pi$ ;
8 end
9 return  $\mathbb{VS}$ ;
```

---

violation formulae (i.e., 81 violation situations) listed in [5]. Based on what is reported in [48], LawBreaker selects 13 traffic laws based on whether they are relevant to the AVs and whether they can be tested using the Apollo + LGSVL framework (e.g., some laws may demand certain road features that are not supported by LGSVL as of now). The violation formulae are automatically generated from the specification of traffic laws specified in the form of STL. They are guaranteed to be correct according to [48]. Our evaluator computes the robustness degree of STL formulae based on RTAMT [40]. The maps we use are all from the map store [30] of LGSVL. For the Car model, we choose LincolnMKZ2017 – the detailed model information can be found online[31].

In the following, we conduct multiple experiments to answer the following Research Questions (RQs). Our experiments are conducted on the four sessions from [7], including *Intersection with Double-Direction Roads (S1)*, *Lane Changing in the Same Road (S2)*, *Intersection with Single-Direction Roads (S3)*, *T-Junction (S4)*. The initial scenarios used in Algorithm 1 come from the existing trace data of LawBreaker. The proxy model in Section 4.2 is implemented as a  $32^*1024$  fully-connected neural network. It also is trained using the existing trace data of LawBreaker. Since LawBreaker is the only one that supports testing against traffic laws, it is the only baseline in our experiments. In view of the randomness, we run LawBreaker and ABLE 4 times (512 runs per time), respectively. For LawBreaker, we select the best testing result as the baseline. For ABLE, we choose the median one for comparison. We set the loop bound  $L$  to 4 and the size of Batch  $B$  to 128 for ABLE. All experiments are conducted on a machine with a 16-Core Intel i7 3.80GHz CPU, an NVIDIA RTX 2080ti GPU, and 32 GB system memory.

### RQ1: How effective is ABLE on generating diverse law-violating scenarios?

To answer this question, we first summarize the number of different law-violating scenarios discovered by LawBreaker and ABLE. Table 2 lists the testing result against Apollo 6 and 7 compared to LawBreaker. Columns **LawBreaker** and **ABLE** show the total number of covered violations in each session on both Apollo versions. Column  $\Delta$  gives the comparison result between LawBreaker and ABLE. Column **Formulae Index** lists the indexes of all the

**Table 2: Comparison with LawBreaker in terms of Covering Violation formulae**

Session	Version	LawBreaker	ABLE	$\Delta$	
				#Violations	Formulae Index
S1	Apollo 6.0	24	28	↑ 4	12, 28, 31, 55
	Apollo 7.0	23	30	↑ 7	12, 28, 31, 32, 33, 40, 55
S2	Apollo 6.0	23	29	↑ 6	2, 40, 41, 46, 73, 77
	Apollo 7.0	26	32	↑ 6	12, 15, 40, 41, 46, 48
S3	Apollo 6.0	26	31	↑ 5	12, 34, 35, 55, 56
	Apollo 7.0	27	36	↑ 9	31, 32, 33, 34, 35, 46, 50, 55, 56
S4	Apollo 6.0	27	29	↑ 2	52, 56
	Apollo 7.0	28	32	↑ 4	31, 46, 55, 56

newly discovered violation formulae. We can find the corresponding violation formulae according to the index in [5]. In total, ABLE totally finds 117 and 130 violation cases in four sessions against 81 violation formula on Apollo 6 and Apollo 7, respectively. ABLE not only covers all the violations covered by LawBreaker, but also finds 17 and 26 more violations than LawBreaker on Apollo 6 and 7, respectively. On average, ABLE offers an increase of 17% and 25% on average compared to LawBreaker. We have made all of these violation publicly available so that they can be used to further improve Apollo.

We then investigate why these scenarios are only covered by ABLE. Our investigation suggests that the reason may be that these newly covered scenarios are indeed hard to cover based on GA. Figure 2 and 3 show the difficulty degree distribution of all the covered violations on Apollo 6 and 7, respectively. For one violation formula, its difficulty degree in one session can be measured by  $1/V_{times}$ , where  $V_{times}$  indicates its the number of times it is violated in the total 512 runs. For example, if there are ten scenarios satisfying violation formula  $\bar{\theta}$  in a session, its difficulty degree is 0.1. Note that a violation formula may have different difficulty degrees in different sessions. On the scatter diagrams, the blue points stand for the violations previously covered by LawBreaker, and yellow points stand for our additional newly-covered violations. It can be found that most of our newly-covered violations have a higher difficulty degree than the previously discovered ones.

Our investigation shows that the difficulty of a formula is mostly related to variables that can be frequently set through certain actions. For instance, the violation formula  $\bar{\theta}_{55}$ , newly covered by ABLE in Session *S1* on Apollo 7.0, has a high difficulty degree of 0.5. It describes a law-violating case that the ego vehicle does not turn on its left-turning signal when turning left. The reason why it is hard to discover in *S1* is that a series of settings are required to make the ego vehicle and an NPC vehicle reach the intersection at the same time, and the ego vehicle has to turn to bypass the NPC vehicle blocked in front. As for  $\bar{\theta}_{63}$ , it describes a law-violating case that the ego vehicle does not turn on the fog light when the fog degree is bigger than or equal to 0.5. It has a low difficulty degree of 0.004 and is easy to be discovered, because the fog remains unchanged once set. Intuitively, this is reasonable as in order to trigger certain law violations, we are required to set the variables accordingly at certain time.

```

1  $\bar{\theta}_{55}$ : Difficulty Degree = 0.5
2 F((direction==Left)and(not(turnSignal==Left)))
3  $\bar{\theta}_{63}$ : Difficulty Degree = 0.004
4 F((fog>=0.5)and(not(fogLightOn==0N)))

```

**Listing 5: The difficulty degree of two formulae in Session S1 on Apollo 7.0**

**Table 3: Complexity and time cost**

Session	SeqL	ComS	Version	G-Cost	T-Cost
S1	43	3.1e+41	6.0	0.39h	7.00h
			7.0	0.42h	7.52h
S2	41	8.0e+42	6.0	0.54h	12.60h
			7.0	0.52h	12.10h
S3	46	4.6e+45	6.0	0.57h	7.83h
			7.0	0.56h	7.80h
S4	82	4.4e+97	6.0	1.22h	8.42h
			7.0	1.30h	9.62h

**Take Away:** ABLE is more effective than the state-of-the-art on generating diverse law-violating scenarios, i.e., covering more distinct ways of violating traffic laws.

**RQ2: How efficient is ABLE?**

To answer the question, we present the details of the training time, and figures showing the number of identified law-violating scenarios over time. Table 3 lists the search space complexity of each session and the time cost in terms of model learning and scenario testing. Column **SeqL** and **ComS** indicate the length of the action sequence for a scenario and the number of sequences in combinational space, respectively. It can be found that the search space in each session is extremely large, and a longer action sequence often means larger search space. Column **G-Cost** and **T-Cost** show the time cost in terms of learning-based scenario generation and scenario testing, respectively. The total execution time of ABLE is Column **T-Cost** + Column **G-Cost**. Except for session *S4*, the learning cost only occupies a small part of the total cost. The reason why *S2* has a higher running costs is that there are many stationary vehicles in *S2*, and as a result, the ego vehicle is often stuck and waits for a long time. Note that we do not list the time cost of LawBreaker, as its **G-Cost** is negligible compared to ABLE and it has the same **T-Cost** with ABLE. Additionally, there are no significant differences in terms of time cost between the two Apollo versions.

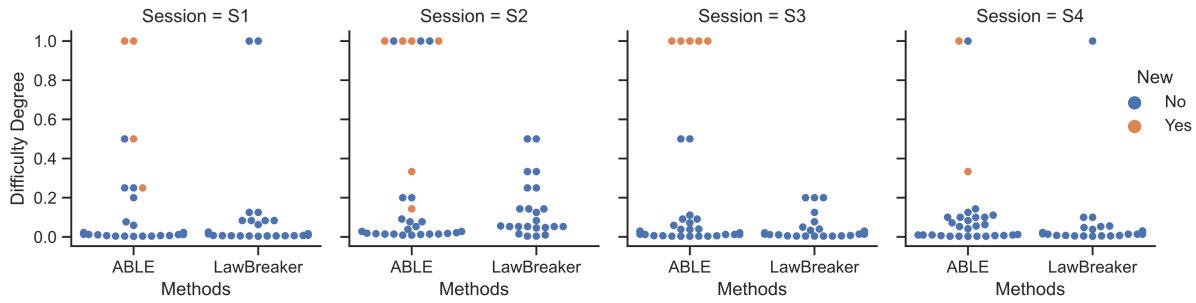


Figure 2: The difficulty degree distribution of covered violations for Apollo 6

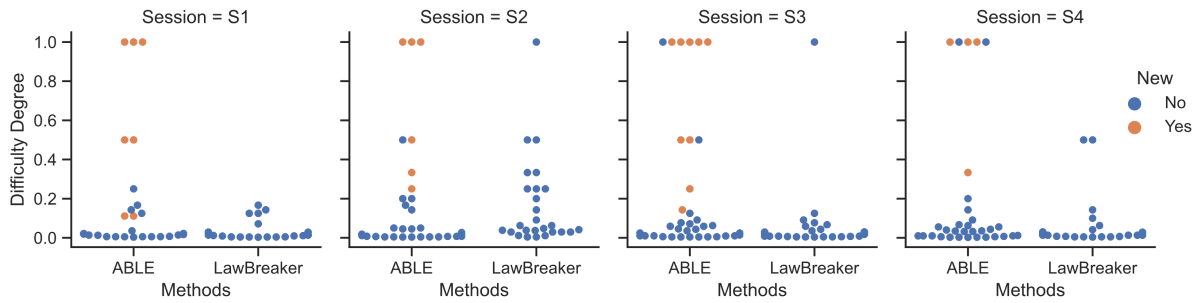


Figure 3: The difficulty degree distribution of covered violations for Apollo 7

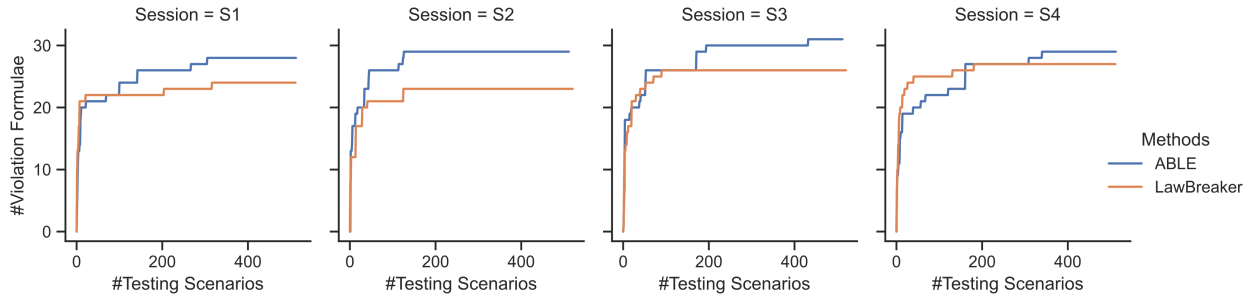


Figure 4: The relation between discovered violations and testing times for Apollo 6

Figure 4 and Figure 5 show the relation between the number of testing scenarios and the number of discovered violations on Apollo 6 and 7, respectively. Our statistic shows that ABLER and LawBreaker find 73% and 90% violation formulae on average in the first 50 testing scenarios, respectively. This almost always corresponds to the easy-to-cover violations with a low difficulty degree in Figure 2 and 3. It can also be found that LawBreaker tends to perform well in the early stage, but contribute little afterwards. It is because the GA in LawBreaker is at high risk of getting stuck in local maxima. Instead, the curves of ABLER continues to rise during the entire stage. It all comes down to the fact that active learning with a dynamic reward function can continuously pour new vitality into the testing.

**Take Away:** 1. The training overhead is minor compared to the testing time. 2. The approach of ABLER (i.e., GFlowNet with dynamic reward function and active learning) is promising as it continues to cover more law-violating scenarios over time.

**RQ3: How effective is active learning and our new reward function (i.e., Optimization 2)?**

To answer the question, we present the details on the comparison between our method (**Active+New**) and two alternative methods (**Inactive+New** and **Active+Max**). Specifically, **Active+New** is an integration of active learning with our new reward function. **Inactive+New** omits active learning from our method, i.e., loop bound  $L$  is set to 1, and the size of Batch  $B$  is set to 512. **Active+Max** replaces our reward function with the maximum

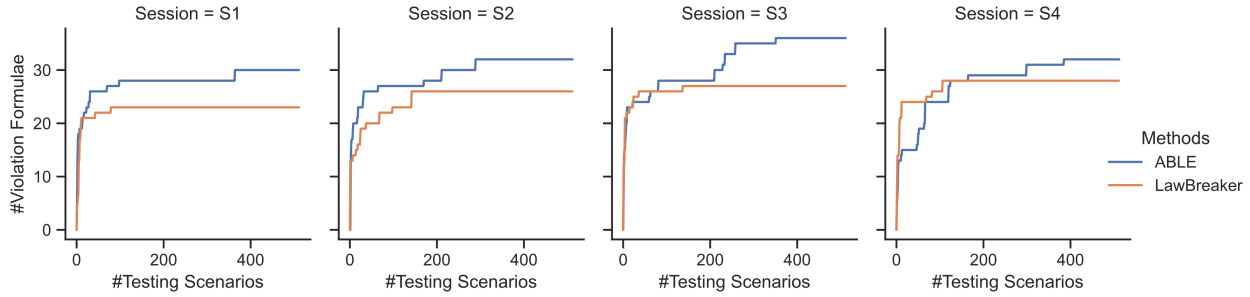


Figure 5: The relation between discovered violations and testing times for Apollo 7

Table 4: Reward function evaluation on Apollo 7

Session	Active+Max	Inactive+New	Active+New	$\Delta$
S1	24	28	30	6/2
S2	30	27	32	2/5
S3	27	24	36	9/12
S4	29	30	32	3/2

robustness which is the one used in LawBreaker, which can be defined as  $\max(\rho(\theta_1, \pi), \dots, \rho(\theta_{81}, \pi))$ . Table 4 shows the comparison result, which is evaluated on Apollo 7.0. Column  $\Delta$  shows the difference between **Inactive+New/Active+Max** and **Active+New** in terms of the number of covered violations. It can be found that our method clearly outperforms the other two methods. Especially, **Inactive+New** is not good as LawBreaker in Session S3. The reason is that **Active+Max** only considers the maximum robustness of a scenario, but ignores the other robustness values, such that part of high-reward scenarios are easily overlooked. As for **Inactive+New**, it only finishes testing a large batch of scenarios without further discovering new law-violating scenarios, since it does not update the reward to re-target the uncovered violation formulae. This shows that changing the reward dynamically is justified in our setting as it helps us to improve the searching efficiency by redirecting search. Note that for optimization 1 and 3, the original design of GFlowNet does not apply and thus it is infeasible to run the experiments with the original GFlowNet. Therefore, we do not conduct the ablation study to demonstrate the effectiveness of optimization 1 and 3.

**Take Away:** Our new reward function combined with the active learning mode can achieve the best effect.

## 6 THREATS TO VALIDITY

First, The effectiveness of ABLE will be effected by the action space of the existing scenario set, e.g., its size and richness. If there is no specific action in the space, such as setting the speed of an NPC to 10.5m/s, then any violations that can only be triggered with this action cannot be discovered by ABLE. It is due to ABLE only creating the new combination modes among the actions in current search space, not creating new actions (i.e., tuning new values for parameters). Therefore, this work dose not analyze whether tuning a parameter for a particular action can make the difference for the

testing. We leave it to future work to incorporate such parameter tuning.

Second, it is hard for us to judge in general whether a scenario is considered realistic since some of them are naturally low-probability events. Nonetheless, it does not mean that ADS gets to ignore them. Based on what we observe, some of the identified scenarios seem realistic to us. For instance, one of the scenarios is that the car does not give way to the moving school bus that first enters into the T-Junction, which breaks the fourth clause in Article 51 of China Traffic Law [21]. This clause stipulates that all vehicles should pass in turn when encountering a release signal (i.e., green traffic light). Its violation formula is the 41-st formula in [5]. As shown in Figure 6-a, the ego vehicle does not give way to the moving school bus when entering the T-Junction. Figure 6-b shows that the ego vehicle continues to move, which eventually leads to a collision. This kind of scene is common in our real world.

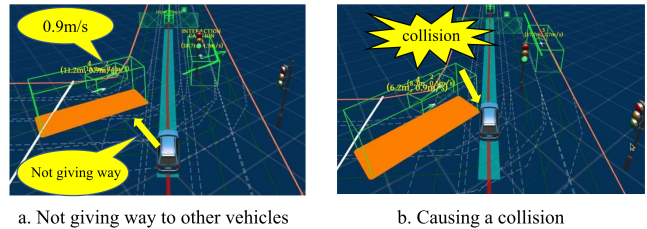


Figure 6: A collision caused by breaking the Article 51 of China Traffic Law

## 7 RELATED WORK

**Testing scenario generation.** Existing works on testing scenario generation mainly explore how to generate critical scenarios that can expose incorrect functionality of ADS. Many works focus on whether a collision occurs or not with evolutionary algorithms[28, 33, 34], Markov decision process[17], rule-based searching[37] and machine learning[23, 27, 32]. For example, AV-Fuzzer[34] optimizes the fuzzing algorithm based on GA under the guide of the distance from other NPC vehicles. Similarly, SceGene [33] leverages the probability model to eliminate traffic scenarios that do not meet the test requirements, and aims to generate diverse scenarios. Jenkins et al. propose an automated approach based on recurrent neural



networks to generate accident scenarios [27]. SAMOTA [23] leverages surrogate models to test ADS with the target of covering many safety requirements, such as avoiding collision with pedestrians. In addition to ‘no-collision’, testing strong oracles only has been studied in a few works [2, 25, 39, 48]. For example, Hungar describes a comprehensive test specification for highway pilots with respect to harmful events-related variables [25]. The most relevant work is LawBreaker [48], which designs a specification language to describe traffic laws and proposes GA-based fuzzing algorithm to find different violations of traffic laws.

Although most existing works propose different methods to generate scenarios for the evaluation of AV, they only focus on simple oracles. Although SAMOTA has the ability of testing many objectives, it can not support testing a much more complicated specification (i.e., traffic laws). As for LawBreaker, its fuzzing algorithm does not work well after the initial 50 tests on average because it shows premature convergence. In this work, we can find diverse scenarios to trigger more traffic violations by combining our new reward function with active learning.

**Object Generation.** Our work is about the problem of learning a sampling policy for generating an object from a sequence of actions. This sampling problem has been studied with numerous generative models [12, 35, 47], Markov Chain Monte Carlo (MCMC) methods [22, 46, 50], reinforcement learning (RL) methods [3, 8, 20] and GFlowNet [11, 26], such as in molecular generation [50] and data imputation [8]. The generative models rely on a given set of positive examples to train the model, thus not taking advantage of the negative examples. RL tends to focus on one or a few dominant modes, i.e., returning maximization objectives. When sampling a batch of diverse objects, MCMC generally only performs local exploration so that it is extremely slow to converge to the target distribution. GFlowNet, first introduced in [11], samples with a probability proportional to the return and then obtains a diverse batch of high-reward objects.

Like the design of *de novo* biological sequences with desired properties [26], diversity is also a key consideration for generating high-reward scenarios in ADS testing. Among these algorithms, only GFlowNet is suitable for our needs. To better adapt to the scenario generation domain, ABLE enhances three optimizations to the vanilla GFlowNet, e.g., dynamically updating reward function and sampling the appropriate type of actions at each sample step.

## 8 CONCLUSION

We propose ABLE, a new ADS testing method based on GFlowNet, to generate diverse law-violating scenarios efficiently. To better adapt to scenario generation, we improve GFlowNet with two optimizations, i.e., dynamically updating objective functions during active learning and only choosing the appropriate type of action at each sample step. We have evaluated ABLE based on Apollo, and LGSVL, and the results show that ABLE outperforms the state-of-the-art by violating 17% and 25% more hard-to-discover laws when testing Apollo 6.0 and Apollo 7.0, respectively. We further conduct experiments to show that our improvements to the vanilla GFlowNet model are relevant and effective.

## ACKNOWLEDGMENT

This research is supported by MoE Academic Research Fund Tier 3 of Singapore (Grant No.: MOET32020-0004), Project of International Cooperation and Exchanges NSFC (Grant No.: 62220106004), Major Research plan of the National Natural Science Foundation of China (Grant No.: 92267204) and Shaanxi Provincial Science and Technology Department Innovation Talent Promotion Program - Science and Technology Innovation Team (Grant No.: 2023-CX-TD-02). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the funding organizations.

## REFERENCES

- [1] Afsoun Afzal, Claire Le Goues, and Christopher Steven Timperley. 2021. Mithra: Anomaly Detection as an Oracle for Cyberphysical Systems. *IEEE Transactions on Software Engineering* (2021), 1–1.
- [2] Matthias Althoff and Sebastian Lutz. 2018. Automatic generation of safety-critical test scenarios for collision avoidance of road vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 1326–1333.
- [3] Christof Angermueller, David Dohan, David Belanger, Ramya Deshpande, Kevin Murphy, and Lucy Colwell. 2019. Model-based reinforcement learning for biological sequence design. In *International conference on learning representations*.
- [4] Anonymous. 2023. Lawbreaker. <https://lawbreaker2022.github.io/>. [Online; accessed Jan 2023].
- [5] Anonymous. 2023. Source code and experiment data. <https://anonymous.4open.science/r/ISSTA2023DATA-0C8E>.
- [6] AVUnit. 2021. AVUnit. <https://avunit.readthedocs.io/en/latest/>. [Online; accessed Apr 2022].
- [7] AVUnit. 2021. Specification-based Autonomous Vehicle Testing. <https://avunit2021.github.io/>. [Online; accessed Agu 2022].
- [8] Philip Bachman and Doina Precup. 2015. Data generation as sequential decision making. *Advances in Neural Information Processing Systems* 28 (2015).
- [9] Baidu. 2019. APOLLO 6.0. <https://github.com/ApolloAuto/apollo/releases/tag/v6.0.0>. [Online; accessed Aug 2022].
- [10] Baidu. 2021. APOLLO 7.0. <https://github.com/ApolloAuto/apollo/releases/tag/v7.0.0>. [Online; accessed Aug 2022].
- [11] Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. 2021. Flow Network based Generative Models for Non-Iterative Diverse Candidate Generation. *Advances in Neural Information Processing Systems* 34 (2021).
- [12] Zihang Dai, Amjad Almahairi, Philip Bachman, Eduard Hovy, and Aaron Courville. 2017. Calibrating energy-based generative adversarial networks. *arXiv preprint arXiv:1702.01691* (2017).
- [13] Steve Dent. 2022. Tesla recalls Full Self Driving feature that lets cars roll through stop signs. <https://techcrunch.com/2022/02/01/tesla-recalls-full-self-driving-feature-that-lets-cars-roll-through-stop-signs/>.
- [14] Jyotirmoy V Deshmukh, Alexandre Donzé, Shromona Ghosh, Xiaoqing Jin, Garvit Juniwala, and Sanjit A Seshia. 2017. Robust online monitoring of signal temporal logic. *Formal Methods in System Design* 51, 1 (2017), 5–30.
- [15] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning (Proceedings of Machine Learning Research, Vol. 78)*, Sergey Levine, Vincent Vanhoucke, and Ken Goldberg (Eds.). PMLR, 1–16.
- [16] R Elvik, A Høye, T Vaa, and M Sørensen. 2009. *The Handbook of Road Safety Measures*, Second 350 Edition.
- [17] Shuo Feng, Xintao Yan, Haowei Sun, Yiheng Feng, and Henry X Liu. 2021. Intelligent driving intelligence test for autonomous vehicles with naturalistic and adversarial environment. *Nature communications* 12, 1 (2021), 1–14.
- [18] Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2019. Scenic: A Language for Scenario Specification and Scene Generation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA) (PLDI 2019). Association for Computing Machinery, New York, NY, USA, 63–78.
- [19] Bernd Gassmann, Fabian Oboril, Cornelius Buerkle, Shuang Liu, Shoumeng Yan, Maria Soledad Elli, Ignacio Alvarez, Naveen Aerrabotu, Suhel Jaber, Peter van Beek, et al. 2019. Towards standardization of AV safety: C++ library for responsibility sensitive safety. In *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2265–2271.
- [20] Sai Krishna Gottipati, Boris Sattarov, Sufeng Niu, Yashaswi Pathak, Haoran Wei, Shengchao Liu, Simon Blackburn, Karam Thomas, Connor Coley, Jian Tang, et al.



2020. Learning to navigate the synthetically accessible chemical space using reinforcement learning. In *International Conference on Machine Learning*. PMLR, 3668–3679.
- [21] Chinese Government. 2021. Regulations for the Implementation of the Road Traffic Safety Law of the People's Republic of China. [http://www.gov.cn/gongbao/content/2004/content\\_62772.htm](http://www.gov.cn/gongbao/content/2004/content_62772.htm). [Online; accessed Mar 2022].
- [22] Will Grathwohl, Kevin Swersky, Milad Hashemi, David Duvenaud, and Chris Maddison. 2021. Oops i took a gradient: Scalable sampling for discrete distributions. In *International Conference on Machine Learning*. PMLR, 3831–3841.
- [23] Fitash Ul Haq, Donghwan Shin, and Lionel Briand. 2022. Efficient online testing for DNN-enabled systems using surrogate-assisted and many-objective optimization. In *Proceedings of the 44th International Conference on Software Engineering*. 811–822.
- [24] Mohammad Hekmatnejad, Shakiba Yaghoubi, Adel Dokhanchi, Heni Ben Amor, Aviral Shrivastava, Lina Karam, and Georgios Fainekos. 2019. Encoding and Monitoring Responsibility Sensitive Safety Rules for Automated Vehicles in Signal Temporal Logic. In *Proceedings of the 17th ACM-IEEE International Conference on Formal Methods and Models for System Design (La Jolla, California) (MEMOCODE '19)*. Association for Computing Machinery, New York, NY, USA, Article 6, 11 pages. <https://doi.org/10.1145/3359986.3361203>
- [25] Hardi Hungar, Frank Köster, and Jens Mazzege. 2017. Test specifications for highly automated driving functions: Highway pilot. *Presentation at Vehicle Test & Development Symposium (2017)*.
- [26] Moksh Jain, Emmanuel Bengio, Alex-Hernandez Garcia, Jarrid Rector-Brooks, Bonaventure FP Dossou, Chanakya Ekbote, Jie Fu, Tianyu Zhang, Micheal Kilgour, Dinghui Zhang, et al. 2022. Biological Sequence Design with GFlowNets. *arXiv preprint arXiv:2203.04115 (2022)*.
- [27] Ian Rhys Jenkins, Ludvig Oliver Gee, Alessia Knauss, Hang Yin, and Jan Schroeder. 2018. Accident scenario generation with recurrent neural networks. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 3340–3345.
- [28] Moritz Klischat and Matthias Althoff. 2019. Generating critical test scenarios for automated vehicles with evolutionary algorithms. In *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2352–2358.
- [29] Mark Koren, Saud Alsaif, Ritchie Lee, and Mykel J. Kochenderfer. 2018. Adaptive Stress Testing for Autonomous Vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*. 1–7.
- [30] LG Silicon Valley Lab. 2019. Maps Content. <https://content.lgsvlsimulator.com/maps/>. [Online; accessed Jul 2022].
- [31] LG Silicon Valley Lab. 2019. Maps Content. <https://content.lgsvlsimulator.com/vehicles/>. [Online; accessed Jul 2022].
- [32] Ritchie Lee, Ole J Mengshoel, Anshu Sakkena, Ryan W Gardner, Daniel Genin, Joshua Silbermann, Michael Owen, and Mykel J Kochenderfer. 2020. Adaptive Stress Testing: Finding Likely Failure Events with Reinforcement Learning. *Journal of Artificial Intelligence Research* 69 (2020), 1165–1201.
- [33] Ao Li, Shitao Chen, Liting Sun, Nanning Zheng, Masayoshi Tomizuka, and Wei Zhan. 2021. SceGene: Bio-Inspired Traffic Scenario Generation for Autonomous Driving Testing. *IEEE Transactions on Intelligent Transportation Systems* (2021).
- [34] Guanpeng Li, Yiran Li, Saurabh Jha, Timothy Tsai, Michael Sullivan, Siva Kumar Sastry Hari, Zbigniew Kalbarczyk, and Ravishankar Iyer. 2020. AV-FUZZER: Finding Safety Violations in Autonomous Driving Systems. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. 25–36.
- [35] Youzhi Luo, Keqiang Yan, and Shuiwang Ji. 2021. Graphdf: A discrete flow model for molecular graph generation. In *International Conference on Machine Learning*. PMLR, 7192–7203.
- [36] Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 152–166.
- [37] Satoshi Masuda, Hiroaki Nakamura, and Kohichi Kajitani. 2018. Rule-based searching for collision test cases of autonomous vehicles simulation. *IET Intelligent Transport Systems* 12, 9 (2018), 1088–1095.
- [38] Alexandra S Mueller, Jessica B Cicchino, and David S Zuby. 2020. What humanlike errors do autonomous vehicles need to avoid to maximize safety? *Journal of safety research* 75 (2020), 310–318.
- [39] Galen E Mullins, Austin G Dress, Paul G Stankiewicz, Jordan D Appler, and Satyandra K Gupta. 2018. Accelerated testing and evaluation of autonomous vehicles via imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 5636–5642.
- [40] Dejan Nicković and Tomoya Yamaguchi. 2020. RTAMT: Online robustness monitors from STL. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 564–571.
- [41] Hari Mohan Pandey, Ankit Chaudhary, and Deepti Mehrotra. 2014. A comparative review of approaches to prevent premature convergence in GA. *Applied Soft Computing* 24 (2014), 1047–1077. <https://doi.org/10.1016/j.asoc.2014.08.025>
- [42] Michael Pines. 2022. Top Causes of Car Accidents: 25 Most Common Causes of Accidents on the Road. <https://seriousaccidents.com/legal-advice/top-causes-of-car-accidents/>. [Online; accessed Agu 2022].
- [43] Vasumathi Raman, Alexandre Donzé, Mehdi Maasoumy, Richard M. Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2014. Model predictive control with signal temporal logic specifications. In *53rd IEEE Conference on Decision and Control*. 81–87. <https://doi.org/10.1109/CDC.2014.7039363>
- [44] Alena Rodionova, Ignacio Alvarez, Maria Soledad Elli, Fabian Oboril, Johannes Quast, and Rahul Mangharam. 2020. How safe is safe enough? Automatic safety constraints boundary estimation for decision-making in automated vehicles. In *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 1457–1464.
- [45] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Martiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, Eugene Agafonov, Tae Hyung Kim, Eric Sterner, Keunhae Ushiroda, Michael Reyes, Dmitry Zelenkovsky, and Seonman Kim. 2020. LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. 1–6.
- [46] Ari Seff, Wenda Zhou, Farhan Damani, Abigail Doyle, and Ryan P Adams. 2019. Discrete object generation with reversible inductive construction. *Advances in Neural Information Processing Systems* 32 (2019).
- [47] Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. 2019. GraphAF: a Flow-based Autoregressive Model for Molecular Graph Generation. In *International Conference on Learning Representations*.
- [48] Yang Sun, Christopher M Poskitt, Jun Sun, Yuqi Chen, and Zijiang Yang. 2022. LawBreaker: An Approach for Specifying Traffic Laws and Fuzzing Autonomous Vehicles. In *37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.
- [49] Jessica Van Brummelen, Marie O'Brien, Dominique Gruyer, and Homayoun Najjaran. 2018. Autonomous vehicle perception: The technology of today and tomorrow. *Transportation Research Part C: Emerging Technologies* 89 (2018), 384–406.
- [50] Yutong Xie, Chence Shi, Hao Zhou, Yuwei Yang, Weinan Zhang, Yong Yu, and Lei Li. 2021. {MARS}: Markov Molecular Sampling for Multi-objective Drug Discovery. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=kHSu4ebxFX>

Received 2023-02-16; accepted 2023-05-03