DynNPC: Finding More Violations Induced by ADS in Simulation Testing via Dynamic NPC Behavior Generation

YOU LU, Fudan University, China
YIFAN TIAN, Fudan University, China
KUN ZHANG, Fudan University, China
DINGJI WANG, Fudan University, China
BIHUAN CHEN*, Fudan University, China
HAOWEN JIANG, Fudan University, China
QICAI CHEN, Fudan University, China
KUN HU, Fudan University, China
XIN PENG, Fudan University, China

Recently, a number of simulation testing approaches have been proposed to generate diverse driving scenarios for autonomous driving systems (ADSs) testing. However, many existing search-based approaches primarily determine NPC behaviors before scenario execution, which limits their ability to model interactions that depend on traffic signals and the Ego vehicle's real-time behavior. As a result, some reported violations may be dominated by unreasonable NPC behaviors, reducing the effectiveness of finding violations induced by the ADS, while the vast search space of NPC behaviors also limits efficiency. To address these limitations, we propose a novel search-based testing framework, DynNPC, to generate more violation scenarios induced by the ADS. Specifically, DynNPC enables NPC vehicles to make maneuver decisions and generate trajectories according to traffic signals and the real-time behavior of the Ego vehicle, using different driving strategies. DynNPC further integrates this dynamic behavior generation with a genetic algorithm-based scenario configuration generator to improve the search for Ego-induced violations. We compare DynNPC with four state-of-the-art scenario-based testing approaches. Our evaluation has demonstrated that DynNPC increases the proportion of violations induced by the ADS among all reported violations, on average, by 125.21%, and improves the number of discovered unique violation patterns induced by the ADS by at least 39.71%. Besides, DynNPC reduces the time to find the first violation induced by the ADS and the average time to find one violation induced by the ADS by 82.13% and 65.70%, respectively. We further conduct ablation studies along with sensitivity analyses of key parameters, and demonstrate the robustness and portability of DynNPC.

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

Additional Key Words and Phrases: Autonomous Driving Systems, Simulation Testing, Scenario-Based Testing

*Bihuan Chen is the corresponding author.

Authors' Contact Information: You Lu, College of Computer Science and Artificial Intelligence, Fudan University, Shanghai, China; Yifan Tian, College of Computer Science and Artificial Intelligence, Fudan University, Shanghai, China; Kun Zhang, College of Computer Science and Artificial Intelligence, Fudan University, Shanghai, China; Dingji Wang, College of Computer Science and Artificial Intelligence, Fudan University, Shanghai, China; Bihuan Chen, College of Computer Science and Artificial Intelligence, Fudan University, Shanghai, China; Haowen Jiang, College of Computer Science and Artificial Intelligence, Fudan University, Shanghai, China; Qicai Chen, College of Computer Science and Artificial Intelligence, Fudan University, Shanghai, China; Kun Hu, College of Computer Science and Artificial Intelligence, Fudan University, Shanghai, China; Xin Peng, College of Computer Science and Artificial Intelligence, Fudan University, Shanghai, China.

ACM 1557-7392/2026/6-ART
https://doi.org/10.1145/3822178

1 Introduction

In recent decades, there has been a significant escalation in both academic and industrial commitment towards the development of autonomous driving systems (ADSs). These systems hold considerable promise for improving road safety, alleviating traffic congestion, and enhancing overall transportation efficiency, thereby driving a transformative shift in the automotive industry [58]. Despite the advancements made by leading companies such as Tesla, Waymo, and Uber, current ADSs still struggle with corner cases and exhibit erroneous behaviors due to the extremely complicated real-world driving environments. These flaws in ADSs can lead to serious consequences and substantial losses, as highlighted by numerous documented traffic incidents [6, 35, 51]. Consequently, extensive testing is needed to ensure the safety and reliability of ADSs before their deployment in the real world.

Leading companies have employed on-road testing to evaluate the performance of ADSs. However, autonomous vehicles have to be driven more than 11 billion miles to demonstrate with 95% confidence that autonomous vehicles are 20% safer than human drivers [34]. This is not only time-consuming but also costly. In contrast, simulation testing offers a more efficient and cost-effective approach to generate diverse and challenging scenarios for ADSs by leveraging the high-fidelity simulators, such as LGSVL [41] and CARLA [20]. These simulators can generate a wide range of scenarios, including various weather conditions, road conditions, and traffic conditions.

Several simulation testing approaches have been proposed for ADSs in simulators, and have been shown to be capable of finding violation scenarios. Some approaches propose domain-specific languages [4, 5, 22, 59, 60] to describe the scenarios, while others generate scenarios by reproducing real-world data [8, 17, 23, 73]. In addition, many search-based approaches [1, 2, 9, 15, 25, 31, 32, 37, 42, 43, 47, 49, 65, 67, 68, 75, 76] have been introduced to generate scenarios guided by different testing objectives. These search-based approaches typically predefine road conditions, weather conditions, and the behaviors of Non-Player Characters (NPCs) (*i.e.,* trajectories of maneuvers at each frame during simulation), and then iteratively mutate the resulting scenario configurations before executing them in the simulator. However, determining NPC behaviors before execution may insufficiently capture interactions conditioned on the Ego vehicle's real-time behavior and traffic signals during simulation, which can lead to unreasonable NPC behaviors. Moreover, exploring the vast search space of NPC behavior mutations is extensive and time-consuming, limiting the efficiency of finding violation scenarios. Recently, several works have explored the use of reinforcement learning to generate interactive and adversarial behaviors of NPCs [19, 61, 63, 71]. While these approaches enable reactive behaviors during execution, they usually rely on learned policies, which require substantial training data and computational resources. Besides, adversarial NPCs are often optimized to maximize the likelihood of violations, which may lead to behaviors that are overly aggressive or difficult to control and interpret.

In fact, while an ADS should behave safely and appropriately even when encountering rule-breaking NPCs, it is unreasonable to expect it to handle completely unreasonable or physically impossible behaviors (*e.g.,* NPCs appearing in unreasonable locations, exhibiting abrupt and extreme speed changes, or violating traffic signals in ways that make collision avoidance impossible). As a result, violations found by scenario-based testing approaches may not necessarily reveal a bug in the ADS under test because the Ego vehicle (*i.e.,* the vehicle controlled by the ADS) may not bear the liability. This is also evidenced by a recent study [32], where 1,109 violation scenarios are automatically generated in 240 hours. After manual diagnosis, all these violations are induced by NPC vehicles. Therefore, improving the effectiveness and efficiency of finding violation scenarios induced by the Ego vehicle, *i.e., Ego-induced violations* (EIVs), remains a challenging problem in simulation testing for ADSs.

In this work, we propose DynNPC, a novel search-based testing framework centered on runtime NPC behavior generation to find more EIVs effectively and efficiently. Specifically, DynNPC enables NPC vehicles to make maneuver decisions and generate trajectories online during simulation execution according to traffic signals and the real-time behavior of the Ego vehicle, under different driving strategies (*i.e.,* the yielding strategy, the adversarial strategy and the overtaking strategy). To support systematic testing, DynNPC further integrates a

genetic algorithm-based generator to produce scenario configurations except for NPC trajectories, and a dedicated scenario executor to ensure the correct execution of generated scenarios.

We have conducted large-scale experiments to evaluate the effectiveness and efficiency of DynNPC. We implement DynNPC based on Apollo 8.0 [7] and LGSVL 2021.3 [41] and compare it with four state-of-the-art scenario-based testing approaches (*i.e.,* AV-Fuzzer [43], AutoFuzz [76], CRISCO [68], and BehAVExplor [15]). Our experiments demonstrate that DynNPC generates the most EIVs in 12 hours among the compared approaches in the straight road and crossroad scenarios, increases the proportion of EIVs among all reported violations, on average, by 125.21%, and improves the number of discovered unique EIV patterns by at least 39.71%. DynNPC also reduces the time to find the first EIV and the average time to find one EIV by 82.13% and 65.70%, respectively. Besides, DynNPC generates smoother and more diverse speed sequences for NPC vehicles. In addition, we analyze the effects of the genetic algorithm-based generator and the three NPC driving strategies on the testing results, study the sensitivity of key maneuver-constraint parameters. Finally, we validate the portability of DynNPC in a more complex roundabout scenario using CARLA.

In summary, this work makes the following main contributions.

- We dynamically generate the behaviors of NPC vehicles using different driving strategies during simulation execution based on traffic signals and the real-time behavior of the Ego vehicle.
- We design and implement DynNPC, a novel search-based testing framework that integrates runtime NPC maneuver decision, online trajectory generation, scenario generation, and scenario execution to improve the possibility of finding Ego-induced violations in ADS simulation testing.
- We conduct experiments with four state-of-the-art scenario-based testing approaches to demonstrate DynNPC's effectiveness and efficiency in finding violation scenarios induced by ADSs.

2 Preliminary and Motivation

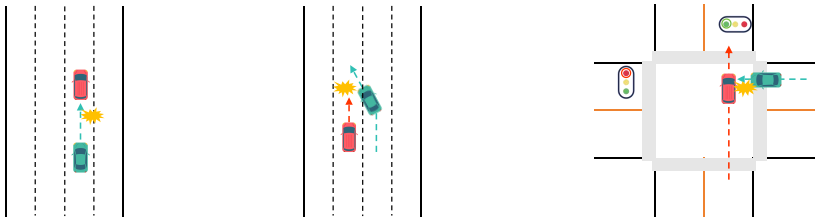
According to [69], a scenario in ADS simulation testing refers to a collection of actors including the Ego vehicle attached with driving tasks, other traffic participants, *i.e.,* Non-Player Characters (NPCs) with behaviors over a period of time, and the environment (*e.g.,* weather conditions and traffic signal configurations). A scenario can be defined as follows.

Definition 1. A scenario $S = \langle t^S, W, E, \mathbb{T}, \mathbb{N} \rangle$ is a 5-tuple where:

- t^S is the maximum allowed frame duration for the scenario.
- $W = \langle rain, fog, wetness, cloudness, time \rangle$ is a tuple used to specify weather and the time of the day. $rain$, fog, $wetness$ and $cloudness$ are float numbers ranging from 0 to 1, and $time$ is an integer between 0 and 23.
- $E = \langle p_{start}, p_{des} \rangle$ is a tuple indicating the driving task of Ego vehicle controlled by ADS under test, consisting of the starting position p_{start} and the destination p_{des}.
- $\mathbb{T} = \{T_0, T_1, \ldots, T_{|\mathbb{T}|-1}\}$ is a set of traffic signal configurations with cooperative and mutually exclusive relationships, mapping the traffic signal T_k to its color $c \in \{RED, YELLOW, GREEN\}$ on a given map.
- $\mathbb{N} = \{N_0, N_1, \ldots, N_{|\mathbb{N}|-1}\}$ is a set of NPCs with behaviors that can be indicated as trajectories in a simulation.

Definition 2. The traffic signal configuration of $T_j = \langle c_{T_j}^{init}, c_{T_j}^{final}, d_{T_j}^{init}, d_{T_j}^{trans}, d_{T_j}^{buffer} \rangle$ is a 5-tuple used to describe the temporal evolution of a traffic signal from its initial color to its final color where:

- $c_{T_j}^{init}$ is the starting signal color, $c_{T_j}^{final}$ is the final signal color after the configured evolution, and $c_{T_j}^{init}, c_{T_j}^{final} \in \{RED, YELLOW, GREEN\}$. $d_{T_j}^{init}$ is the duration that the signal displays the color $c_{T_j}^{init}$.
- $d_{T_j}^{trans}$ is the duration of the transition phase, if needed, when the signal evolves from $GREEN$ to RED, it displays $YELLOW$ for $d_{T_j}^{trans}$ to warn vehicles before turning RED [39].



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

Fig. 1. Examples of Violations Induced by NPCs

- $d_{T_j}^{buffer}$ is the safety buffer duration, if needed, during which the signal remains *RED* before switching to *GREEN*, allowing vehicles that have already entered the intersection to clear it [39].

Definition 3. The trajectory of the Ego vehicle E or an NPC $N_k \in \mathbb{N}$ is a 2-tuple $\langle P, V \rangle$, where:

- $P = \langle p^0, p^1, \ldots, p^{d-1} \rangle$ is a sequence of waypoints that the traffic participant follows at each timestamp during the frame duration d. A waypoint p indicates a specific location on the map in the coordinate system.
- $V = \langle v^0, v^1, \ldots, v^{d-1} \rangle$ is a sequence of speed of the traffic participant at each timestamp during the frame duration d.

Scenario-based testing [77] configures the scenarios and executes them in a simulator bridged with the ADS under test. Then, the search problem for violation scenarios can be cast to a (multi-objective) optimization problem guided by the objective function that formalizes a specific search goal, such as the minimal distance between vehicles and minimal *time-to-collision* [67]. Previous approaches mutate the scenario configurations iteratively to change the weather conditions W, driving tasks of the Ego vehicle E, trajectories of NPCs $\mathbb{N}$ [37, 43], and traffic signal configurations $\mathbb{T}$ [32] in a vast searching space. Besides, some works [67, 68] compose multiple trajectories of NPCs as behavior patterns to form the scenario configuration and choose to mutate these patterns, compressing the search space to improve efficiency. All of these approaches share the common feature of generating NPC behaviors in the scenario configuration prior to simulation execution. These approaches have been demonstrated to be capable of finding violation scenarios. However, the ADS may not bear the liability for the violations because the violations may be induced by the NPCs.

We introduce the following motivating examples to demonstrate the limitation. We use the red car to represent the Ego vehicle and green car to represent the NPC vehicle. As shown in Fig. 1a, the NPC vehicle hits the stopped Ego vehicle from behind. According to the Uniform Vehicle Code (UVC) [57], the rear vehicle is generally responsible in rear-end collisions. In Fig. 1b, the NPC vehicle suddenly changes lanes with an abrupt speed change over a short distance in front of the Ego vehicle and the Ego vehicle decelerates but still fails to avoid the collision. In Fig. 1c, the Ego vehicle moves forward because it recognizes that the traffic signal of its current lane is green and the intersection is clear. When the Ego vehicle passes the intersection, it is collided by a high-speed NPC vehicle that does not obey the red signal in the horizontal direction. In fact, all the aforementioned violations reported by previous approaches are induced by the unreasonable NPC behaviors. These approaches ignore the behavior of the Ego vehicle during scenario execution and inevitably introduce unreasonable NPC behaviors generated through iterative mutations prior to execution, resulting in a high false positive rate of reported violations in the testing results.

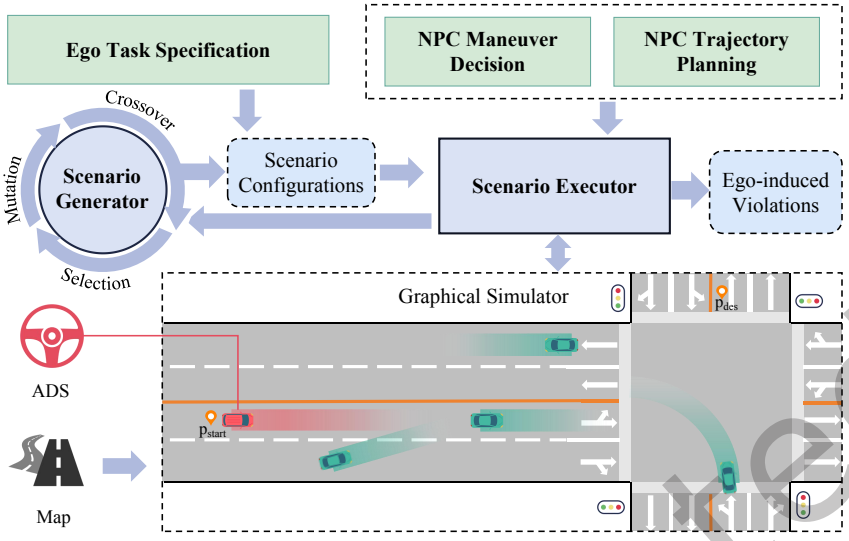


Fig. 2. Approach Overview of DynNPC

To address the above limitation while facilitating the search for EIVs, we propose DynNPC to dynamically generate the behaviors of NPC vehicles based on traffic signals and the real-time behavior of the Ego vehicle during simulation execution, finding more EIVs effectively and efficiently.

3 Methodology

During the simulation execution, NPC vehicles can have a variety of behaviors, and different driving maneuvers may have different degrees of impact on the Ego vehicle. Besides, NPC vehicles should obey traffic signals when taking maneuvers at intersections equipped with traffic lights. Even under the same maneuver, the generated trajectories may still differ in their speed profiles. For example, when an NPC vehicle changes lanes in front of the Ego vehicle, it may either slow down to yield before completing the lane change or speed up to overtake in front of the Ego vehicle. It may adopt an adversarial driving strategy, posing a greater threat to the Ego vehicle. Therefore, the key challenge for dynamically generating the behaviors of NPC vehicles during simulation execution lies in *(1) how to determine the maneuvers of the NPC vehicles and (2) how to generate the trajectories obeying traffic signals using different driving strategies.*

Fig. 2 shows the overview of DynNPC. First, DynNPC specifies the driving task of the Ego vehicle (see Sec. 3.1). Then, it enables NPC vehicles to make maneuver decisions based on the real-time behavior of the Ego vehicle (see Sec. 3.2) and to generate trajectories under different driving strategies while obeying traffic signals (see Sec. 3.3). To improve the effectiveness and efficiency of finding EIVs, DynNPC connects the ADS with the simulator and selects a road segment from the map loaded in the simulator. It further employs a genetic algorithm-based scenario generator (see Sec. 3.4) to automatically search scenario-level configurations in our genetic representation. Finally, DynNPC implements a new scenario executor (see Sec. 3.5) to ensure correct scenario execution and collect testing results.

3.1 Ego Task Specification

As mentioned in Sec. 2, the driving task of the Ego vehicle can be represented by the starting position p_{start} and the destination p_{des}. The goal of the Ego vehicle is to navigate from p_{start} to p_{des} while considering dynamic traffic

conditions and adhering to traffic signals. We select specific road segments from the provided map, supporting both straight roads and crossroads controlled by traffic lights. A set of p_{start} and p_{des} extracted from the map are assigned to ensure a diverse range of driving tasks. For straight road scenarios, multiple NPC vehicles are initially distributed within the road segment between p_{start} and p_{des}. These NPC vehicles may occupy different lanes and follow varying speeds, or perform lane changes during simulation execution, creating a dynamic environment for the Ego vehicle. The Ego's task is to pass through this region, ensuring safe interactions with NPC vehicles while reaching p_{des}.

For crossroad scenarios, NPC vehicles are initially positioned in designated waiting areas before the stop lines at the crossroad. The Ego vehicle is required to navigate the crossroad by either going straight, turning left, or turning right, based on the assigned p_{start} and p_{des}. During this process, the Ego vehicle must strictly comply with traffic signals and avoid potential conflicts with crossing NPC vehicles.

3.2 NPC Maneuver Decision

According to the data provided by National Highway Traffic Safety Administration (NHTSA) [3], in real-word driving scenarios, vehicles can perform various maneuvers including decelerating, accelerating, lane changing, turning left, turning right, parking, and backing up, [12, 54]. We introduce the constraints of maneuvers we adopt and illustrate how NPC vehicles in DynNPC dynamically determine their driving maneuvers based on the behavior of the Ego vehicle.

Maneuver Constraints. DynNPC adopts a maneuver taxonomy based on CRISCO [68], but our maneuver constraints are not directly reused from CRISCO. CRISCO mainly constrains the geometric and topological feasibility of predefined behavior patterns, such as the relative positions of start and destination points, lane relationships, and movement directions. In contrast, DynNPC introduces additional runtime-oriented constraints for maneuver feasibility, including the longitudinal safety distance to the Ego vehicle, the activation of brake/turn signals, solid-line checking for lane changes, lane-type constraints for turning maneuvers, traffic-signal compliance, and speed-related constraints consistent with the ADS under test. These constraints are introduced to ensure that NPC vehicles do not appear in unreasonable positions, perform abrupt lane changes within short distances that could lead to unavoidable collisions, and comply with traffic signals so that their behaviors remain as reasonable as possible. By enforcing such constraints during runtime maneuver decision, we aim to reduce unreasonably designed scenarios where collisions are primarily caused by improper NPC behaviors rather than by the Ego vehicle.

Decelerating. When the NPC vehicle N_k decides to decelerate at frame t_0, it must ensure a safe deceleration longitudinal distance to the Ego vehicle behind if they are in the same lane, and activate the brake lights, avoiding sudden braking and rear-end collision. The specification can be defined by Eq. 1,

$$D_{N2E}(p_{N_k}^{t_0}, p_E^{t_0}) \geq threshold \wedge \ brakeSignal(p_{N_k}^{t_0}) = True \tag{1}$$

where $D_{N2E}(p_{N_k}^{t_0}, p_E^{t_0})$ returns the distance between the NPC vehicle and the Ego vehicle at frame t_0 when they drive in the same lane, and *threshold* is set to 30 meters by default as a conservative safety threshold. The function $brakeSignal(p_{N_k}^{t_0})$ returns the status of NPC vehicle's braking signal. Besides, the maximum deceleration rate of the NPC vehicle is set consistently with the configuration of the ADS under test, *e.g.,* 8 m/s^2 of Apollo [7].

Accelerating. When the NPC vehicle N_k decides to accelerate at frame t_0, if the NPC vehicle is behind the Ego vehicle in the same lane, it must maintain a safe longitudinal distance from the Ego vehicle. In this case, the maximum speed of the NPC vehicle after acceleration must not exceed the speed of the Ego vehicle, so as to avoid introducing an unreasonable rear-end collision. Besides, it should ensure that the speed after acceleration

does not exceed the speed limit of the current road section. The specification can be defined by Eq. 2,

$$\left(0 < D_{N2E}(p_E^{t_0}, p_{N_k}^{t_0}) \leq threshold \Rightarrow MaxSpeed_{N_k} \leq v_E^{t_0}\right) \wedge \ MaxSpeed_{N_k} \leq \text{speedLimit} \tag{2}$$

where $D_{N2E}(p_E^{t_0}, p_{N_k}^{t_0})$ returns the distance between the Ego vehicle and the NPC vehicle at frame t_0 when they drive in the same lane, *threshold* is set to 30 meters by default as a conservative safety threshold, and speedLimit is extracted from the map. Besides, the maximum acceleration rate of the NPC vehicle is also set consistently with the configuration of the ADS under test, *e.g.,* 8 m/s^2 of Apollo [7].

Lane Changing. When the NPC vehicle N_k is going to take lane changing maneuver to the left at frame t_0, it must ensure a safe lane changing distance from the following Ego vehicle, activate the left turn signal when changing lanes, and avoid performing the maneuver within a solid-line area. The specification can be defined by Eq. 3,

$$D_{N2E}(p_{N_k}^{t_0}, p_E^{t_0}) \geq threshold \wedge \ leftSignal(p_{N_k}^{t_0}) = True \wedge \ isSolid(p_{N_k}^{t_0}) = False \tag{3}$$

where $D_{N2E}(p_{N_k}^{t_0}, p_E^{t_0})$ returns the longitudinal distance between the NPC vehicle and the Ego vehicle at frame t_0, and *threshold* is set to 30 meters by default as a conservative safety threshold for lane-changing feasibility. The function $leftSignal(p_{N_k}^{t_0})$ returns the status of NPC vehicle's left turn signal, while $isSolid(p_{N_k}^{t_0})$ indicates whether the surrounding lane markings of the NPC vehicle at frame t_0 are solid lines.

Similarly, when NPC N_k performs a right lane change at frame t_0, the specification is defined by Eq. 4.

$$D_{N2E}(p_{N_k}^{t_0}, p_E^{t_0}) \geq threshold \wedge \ rightSignal(p_{N_k}^{t_0}) = True \wedge \ isSolid(p_{N_k}^{t_0}) = False \tag{4}$$

where $rightSignal(p_{N_k}^{t_0})$ checks if the right turn signal is turned on.

Turning Left. When the NPC vehicle N_k decides to turn left at an intersection at frame t_0, it must ensure that the maneuver is performed safely in the left turn lane, following traffic signals (see Speed Planning). Besides, it must activate the left turn signal before initiating the turn. The specification can be defined by Eq. 5,

$$isLeftTurnLane(p_{N_k}^{t_0}) = True \wedge leftSignal(p_{N_k}^{t_0}) = True \wedge RedSignalObeying = True \tag{5}$$

where $isLeftTurnLane(p_{N_k}^{t_0})$ ensures that the NPC vehicle only turns left in the left turn lane and $leftSignal(p_{N_k}^{t_0})$ ensures the activation of the left turn signal.

Turning Right. When the NPC vehicle N_k decides to turn right at an intersection at frame t_0, it must ensure that the maneuver is performed safely in the right turn lane and activate the right turn signal before initiating the turn. In our setting, when no dedicated right-turn signal is present, a right turn is treated as a permissible maneuver even if the forward through signal is red according to traffic regulations [13, 21, 56]. The specification can be defined by Eq. 6,

$$isRightTurnLane(p_{N_k}^{t_0}) = True \wedge \ rightSignal(p_{N_k}^{t_0}) = True \tag{6}$$

where $isRightTurnLane(p_{N_k}^{t_0})$ ensures that the NPC vehicle only turns right in the right turn lane and $rightSignal(p_{N_k}^{t_0})$ ensures the activation of the right turn signal.

Parking. When the NPC vehicle chooses to decelerate to stop, it must ensure that the parking maneuver is performed in a designated parking area, at a complete stop at frame t_n. The specification can be defined by Eq. 7,

$$prohibitParkingZone(p_{N_k}^{t_n}) = False \wedge \ v_{N_k}^{t_n} = 0 \tag{7}$$

where $prohibitParkingZone(p_{N_k}^{t_n})$ returns whether the NPC vehicle is stopping in prohibited areas (*e.g.,* intersections).

Backing up. We do not consider backing up maneuvers because it is the most dangerous maneuver in highways and is forbidden in the crossroad.

Maneuver Decision. For the NPC vehicle N_k, we determine its maneuver at frame t based on the behavior of the Ego vehicle. Specifically, we obtain the speed v_E^t of the Ego vehicle at frame t from simulator, and its driving task (*i.e.,* p_{start} and p_{des}). Then, we generate the expected trajectory of the Ego vehicle. On the straight road, the expected trajectory is generated according to the Ego vehicle's heading direction provided by the simulator, assuming uniform motion at the current speed v_E^t, resulting in a straight trajectory for the next period of time. On the crossroad, the expected trajectory is generated according to the driving task, forming a path that connects p_{start} and p_{des} while adhering to roadway centerline. Given the position $p_{N_k}^t$ of the NPC vehicle obtained from the simulator, we identify all feasible driving maneuvers that satisfy the maneuver constraints. Among these maneuvers, if there exists a sequence of waypoints generated by maneuver m (see Sec. 3.3) that overlaps with the expected trajectory of the Ego vehicle, the NPC vehicle will execute maneuver m in the subsequent time period to enhance its interaction with the Ego vehicle, thereby reducing the search space for EIVs. If none of the feasible maneuvers' waypoint sequences overlap with the Ego's expected trajectory, the NPC vehicle will randomly select one from the set of all available maneuvers.

Fig. 3 shows an example of maneuver decision on the crossroad. According to the driving task, the Ego vehicle will proceed straight through the crossroad when its traffic signal is green. At this moment, the NPC vehicle A, positioned in the oncoming through and left-turn lane, can choose to either go straight or turn left. It will select the left-turn maneuver from the available options, as the trajectory of left-turn maneuver overlaps with the expected trajectory of the Ego vehicle, increasing the likelihood of interaction between the two vehicles. Meanwhile, the NPC vehicle B, located in the cross-direction through and right-turn lane, faces a red signal for through traffic due to the mutually exclusive signal configuration with the Ego vehicle's green signal. Under this configuration, NPC vehicle B is permitted to execute only a right turn maneuver.

3.3 NPC Trajectory Planning

As mentioned in Sec. 2, a trajectory consists of a sequence of waypoints and its speed sequence. Thus, we divide trajectory planning process into two tasks, namely waypoint generation and speed generation.

Waypoint Generation. The waypoint generation task is responsible for generating a sequence of waypoints that the NPC vehicle should follow to execute the selected maneuver. Similar to previous work [60, 68], Bézier curves [53], which are parametric curves providing a smooth transition, are used to calculate the waypoints for this phase. A Bézier curve $B(t)$ can be constructed by four control points $P_0 - P_3$, i.e., $B(\zeta) = (1-\zeta)^3 P_0 + 3(1-\zeta)^2\zeta P_1 + 3(1-\zeta)\zeta^2 P_2 + \zeta^3 P_3$, $\zeta \in [0, 1]$. For example, when generating the waypoints of lane changing maneuver, we set the current position $p_{N_k}^{t_0}$ of the NPC vehicle as P_0, and the target lane position extracted from map as P_3. Then, we compute the two intermediate control points P_1 and P_2 according to the lane geometry. Specifically, let $\vec{h}_0$ and $\vec{h}_3$ denote the heading directions of the current lane at P_0 and the target lane at P_3, respectively. We place P_1 by moving forward from P_0 along $\vec{h}_0$, and place P_2 by moving backward from P_3 along $\vec{h}_3$, using the same longitudinal offset proportional (0.3×) encoded in the waypoint specification [68] to the distance between P_0 and P_3. In this way, the Bézier curve follows the entering direction of the current lane and the exiting direction of the target lane, yielding a smooth and plausible lane-changing trajectory.

Speed Generation. The speed generation task is responsible for assigning speeds to the generated waypoints. By default, NPC vehicles operate at their initially assigned speeds. However, strictly maintaining this speed may lead to running red lights on the crossroad. Additionally, when interacting with the surrounding Ego vehicle, adhering rigidly to the default speed may result in suicidal behaviors, potentially leading to aggressive collisions. Thus, we use the Station-Time graph (*i.e., s-t* graph) [50] to modify the speed profile of the NPC vehicle's generated waypoints. The *s-t* graph has also been used in prior work such as ACAV [64], where it is adopted to analyze

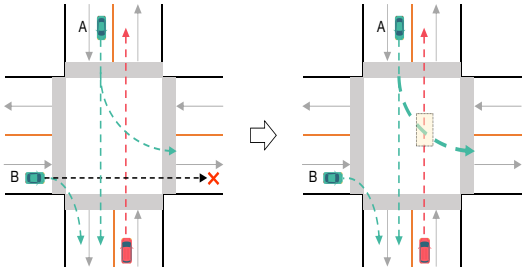


Fig. 3. Examples of NPC Maneuver Decision on Crossroad

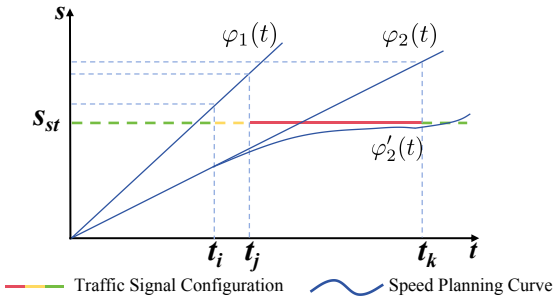


Fig. 4. Traffic Signal Obeying with *s-t* Graph

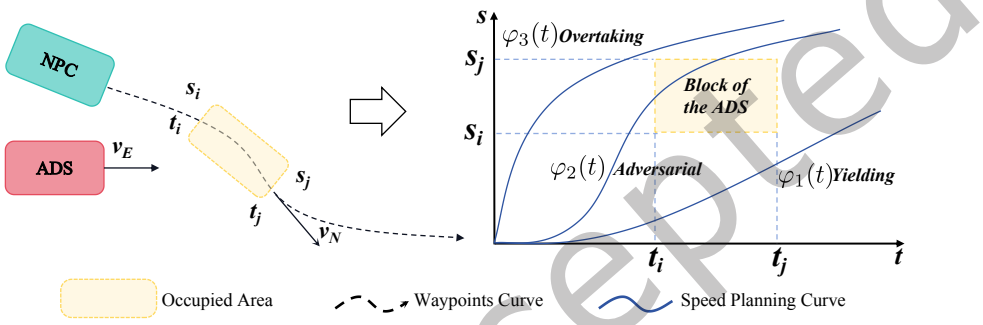


Fig. 5. Driving Strategy Implementation with the *s-t* Graph

ADS planning states in accident recordings. In DynNPC, we use the *s-t* graph for runtime speed planning of NPC vehicles. Specifically, in an *s-t* graph, time is the horizontal axis, the planned longitudinal trajectory distance is the vertical axis, and the planned longitudinal trajectory is a curve. Each point on the curve represents a waypoint on the planned trajectory, and the curve's gradient represents the speed. Based on the generated waypoints, we use the *s-t* graph to adjust the NPC vehicle's speed profile, so as to enable NPC vehicles to obey traffic signals and implement different driving strategies when their expected trajectories overlap with that of the Ego vehicle.

Traffic Signal Obeying. When the NPC vehicle N_k attempts to pass through the crossroad at frame t with an initial speed $v^t_{N_k}$, we retrieve from the simulator both the NPC vehicle's current distance to the stop line, denoted as s_{st}, and the traffic signal configuration. As illustrated in Fig. 4, assuming that the traffic signal remains green from t to t_i, turns yellow from t_i to t_j, and becomes red from t_j to t_k, we use $v^t_{N_k}$ as the slope to draw the speed planning curve of N_k. Here, $\varphi_1(t)$ means that N_k can enter the crossroad before t_i (*i.e.,* green signal) at the current speed. Conversely, $\varphi_2(t)$ means that the vehicle will pass the stop line after t_j, which will result in running a red light. To prevent this, we replan the speed profile of N_k, gradually decelerating until N_k comes to a complete stop before the stop line (*i.e.,* the $\varphi_2'(t)$ curve). Finally, we update the speed of each waypoint based on the gradient of the newly generated speed planning curve, ensuring that the NPC vehicle adheres to traffic signals while executing its maneuver.

Driving Strategy Implementation. When the waypoints of the NPC vehicle N_k overlap with the expected trajectory of the Ego vehicle after frame t, we use the *s-t* graph to generate the speed of N_k, as shown in Fig. 5. Assuming that the total length of the waypoint sequence generated by N_k's maneuver m is s, and the Ego vehicle will travel at a constant speed v^t_E obtained from the simulator since frame t, then, the Ego vehicle will occupy the

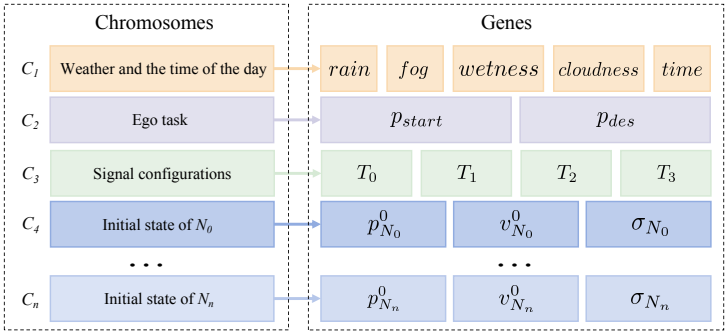


Fig. 6. Genetic representation used by DynNPC

s_i to s_j section of N_k's waypoints from time t_i to t_j. Therefore, there is a block of the ADS in the *s-t* graph where a collision is most likely to occur. Then, we can plan the speed of N_k with three different driving strategies as follows.

- **Yielding Strategy:** The NPC vehicle decelerates and yields to the Ego vehicle, planning a speed profile below the block of the ADS (*e.g.,* $\varphi_1(t)$ in Fig. 5).
- **Adversarial Strategy:** The NPC vehicle travels with adversarial speeds, planning a speed profile through the block of the ADS (*e.g.,* $\varphi_2(t)$ in Fig. 5).
- **Overtaking Strategy:** The NPC vehicle accelerates to overtake the Ego vehicle, planning a speed profile above the block of the ADS (*e.g.,* $\varphi_3(t)$ in Fig. 5).

Finally, we calculate the gradient of the speed planning curve and update the speed of each waypoint of N_k to implement the selected driving strategy, improving the diversity of NPC behaviors.

3.4 Scenario Generator

To systematically explore scenario-level configurations beyond runtime NPC trajectory generation, DynNPC adopts a genetic algorithm-based scenario generator similar in spirit to previous work [2, 37, 43, 67, 68] to automatically generate scenario configurations for the ADS under test. We elaborate on the design of the genetic representation of the scenario configurations and its search operators.

Genetic Representation. We use the definition of a scenario in Sec. 2 to configure scenarios. However, instead of explicitly specifying the trajectories of NPC vehicles in our scenario configurations *Conf*, we assign them driving strategies (*i.e.,* yielding strategy, adversarial strategy, or overtaking strategy). Fig. 6 illustrates a genetic representation of an individual generated by DynNPC. Each individual, which is a scenario configuration in our representation, consists of several chromosomes. C_1 indicates the weather conditions and the time of the day (*i.e.,* W), C_2 indicates the driving task of the Ego vehicle (*i.e.,* E), and C_3 indicates the traffic signal configurations at the crossroad (*i.e.,* $\mathbb{T}$). Besides, we use chromosomes from C_4 to C_n that have 3 genes (*i.e.,* the initial position $p^0_{N_k}$, the initial speed $v^0_{N_k}$ and the driving strategy σ_{N_k} adopted by N_k) to indicate the NPC vehicles engaged in the scenario (*i.e.,* $\mathbb{N}$), respectively. This representation differs from prior genetic algorithm-based approaches in that it does not encode full NPC trajectories as genes. Instead, it only encodes scenario-level initial conditions and high-level driving strategies, while the detailed trajectories are generated online by the maneuver decision and trajectory planning modules during simulation execution.

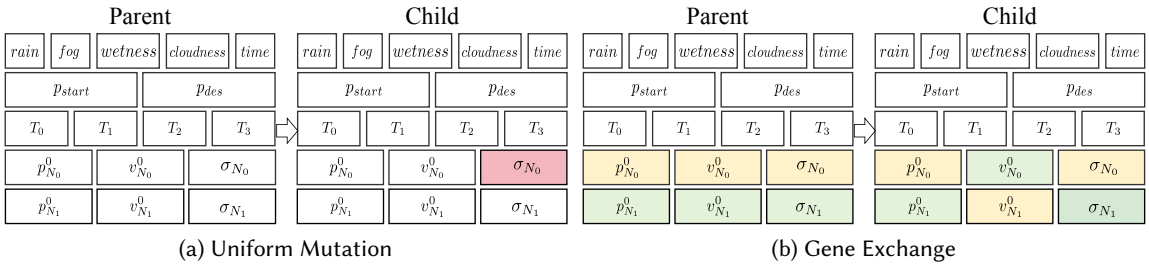


Fig. 7. Examples of Mutation

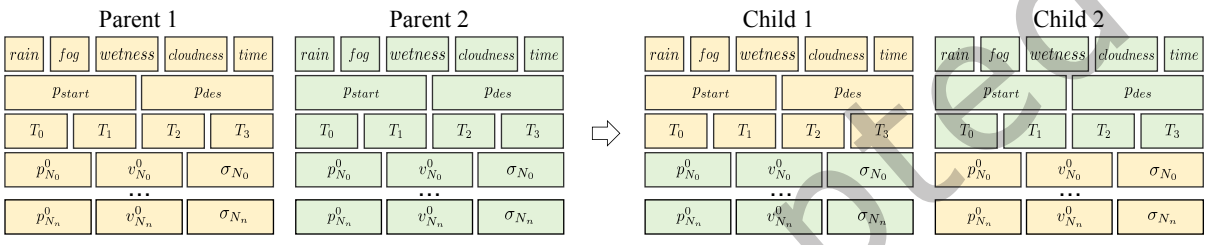


Fig. 8. An Example of Crossover

Mutation. Mutation introduces intra-individual variation in our scenario representation. As shown in Fig. 7, we mainly apply uniform mutation to randomly change one gene in the chromosomes, and additionally exchange the same gene between two NPC chromosomes within one individual.

Crossover. Crossover introduces inter-individual variation through single-point crossover on chromosomes between two individuals. As shown in Fig. 8, a random crossover point is selected, and the chromosome segments after that point are exchanged to generate new offspring.

Selection. For seed selection, DynNPC reuses the energy-based evaluation and selection framework of BehAVExplor [15], but adapts it to our scenario representation and testing objective. In BehAVExplor, the energy mechanism is used to balance violation feedback and behavior diversity during seed selection. In DynNPC, we use the same mechanism to prioritize scenario configurations that are more likely to expose EIVs.

Violation Feedback. We calculate the violation score v_I of a seed scenario configuration I by Eq. 8,

$$v_I = \sum_{v \in \{collision, Lines, P_{des}\}} f_v(I) \tag{8}$$

where I denotes an executed scenario configuration, and $f_v(I)$ is the violation feedback function for the corresponding specification. Specifically, $f_{collision}(I)$ returns the minimum distance between the Ego vehicle and NPC vehicles during the execution of I; once an actual collision occurs, the value becomes zero. Similarly, $f_{Lines}(I)$ returns the minimum distance between the Ego vehicle and illegal lines *Lines*; once the Ego vehicle hits an illegal line, the value becomes zero. Following [15], we define $f_{P_{des}}(I)$ as $f_{P_{des}}(I) = \max(10 - D_{E2des}(p^n_E, p_{des}), 0)$, where $D_{E2des}(p^n_E, p_{des})$ denotes the distance between the final position of the Ego vehicle p^n_E and its destination p_{des} after simulation. This definition means that once the Ego vehicle is still more than 10 meters away from the destination at the end of the simulation, the corresponding violation feedback becomes zero, indicating that the execution is close to violating the destination-reaching requirement. Therefore, lower values of v_I generally indicate that the corresponding scenario configuration is closer to exposing one or more violations. We use the

unweighted sum of these three terms as a unified violation-feedback objective for seed evaluation during search, rather than to model the relative severity of different violations.

Behavior Diversity. We calculate a minimum distance $d_{I'}$ to measure the diversity of a new mutant I' by Eq. 9,

$$d_{I'} = \min_{i \in Q} dis(\mathbf{H}_{I'}, \mathbf{H}_i) \tag{9}$$

where Q is the current seed corpus, $\mathbf{H}_I$ denotes the behavior feature representation of the Ego vehicle in seed I, extracted from the observed Ego trajectory after executing I, and $\mathbf{H}_i$ denotes the behavior feature representation of the Ego vehicle in seed $i \in Q$. The function $dis(\mathbf{H}_{I'}, \mathbf{H}_i)$ returns the Hamming distance [27], which measures the behavioral diversity of the Ego vehicle between two seeds. The larger $d_{I'}$ is, the more diverse the Ego behavior in I'.

Energy-Based Selection. When the fuzzer starts, all initial seeds generated randomly are assigned with the same energy (*i.e.,* 1). Then, we update the energy E_I of seed scenario configuration I and calculate the energy $E_{I'}$ for the newly mutated scenario configuration I' during fuzzing, where I' is mutated from I.

Following [15], we update the energy of I by Eq. 10,

$$E_I = E_I + w_1 \cdot \Delta E_F + w_2 \cdot \Delta E_V + w_3 \cdot \Delta E_S \tag{10}$$

where w_1, w_2 and w_3 are parameters that can adjust the weights of the three factors. We use the same values as [15] to set the parameters. ΔE_F represents the failure frequency. For the seed I, we record the number of failed/non-failed test cases (denoted as #F and #NF) that are mutated from I, and the ΔE_F is calculated by Eq. 11,

$$\Delta E_F = \begin{cases} \#\mathrm{F}/\#\mathrm{NF} + \#\mathrm{F}, & I' \text{ is a failed test case} \\ -\#\mathrm{NF} \cdot 0.1/(\#\mathrm{NF} + \#\mathrm{F}), & I' \text{ is a benign test case} \end{cases} \tag{11}$$

ΔE_V represents the violation degree change. As it may be hard to directly generate failed test cases, we consider the violation degree change between the seed I and the newly mutated scenario configuration I', which is calculated by Eq. 12,

$$\Delta E_V = \frac{v_I - v_{I'}}{1 - d_{I'} + \gamma} \tag{12}$$

where γ is a small value (*e.g.,* 10^{-5}) to avoid zero denominator. ΔE_S is the selection frequency, which is a fixed energy decay value following [15], *i.e.,* ΔE_S = -0.05.

With respect to the newly generated I', we assign the initial energy $E_{I'}$ by Eq. 13, where ΔE_V is the violation degree change defined in Eq. 12, and w_2 is its corresponding weight in Eq. 10.

$$E_{I'} = 1 + w_2 \cdot \Delta E_V \tag{13}$$

Based on the energy, we compute a selection probability of each seed I by Eq. 14,

$$p_I = \frac{\max(E_I, 0)}{\sum_{i \in Q} E_i} \tag{14}$$

where Q is the current seed corpus, E_I is the energy of seed I, and E_i is the energy of seed i in Q. We use the max function to make sure all probabilities are nonnegative. Finally, the seed with higher energy will have a higher probability to be selected for breeding the next generation.

3.5 Scenario Executor

Since the scenario configuration generated in Sec. 3.4 does not include the NPC vehicle trajectories, we cannot directly utilize the scenario executor from previous works [15, 28, 37, 43, 67, 68, 76]. Moreover, each NPC vehicle in the scenario requires asynchronous behavior generation. We implement a new scenario executor capable of

Algorithm 1: Scenario Execution

Input: the map: Map, the maximum frame duration: t^S, and the configuration of scenario: $Conf$
Output: the record of the simulation: $Record$

```
1  NPClist ← InitNPC(Conf.ℕ);
2  sim ← Initialize(Map, Conf, NPClist);
3  foreach NPC ∈ NPClist do
4  |   NPC.status ← IDLE;
5  end
6  for t ← 1 to t^S do
7  |   foreach NPC ∈ NPClist do
8  |   |   if NPC.status = IDLE then
9  |   |   |   m ← NPC.DecideManeuver(sim.Ego);
10 |   |   |   m.trajectory ← NPC.PlanTrajectory(m, sim.Ego);
11 |   |   |   NPC.status ← RUNNING;
12 |   |   |   MonitorSignal(NPC, m);
13 |   |   end
14 |   end
15 |   sim.run(0.1);
16 end
17 Record ← UpdateRecord(sim.Ego, NPClist);
18 return Record;
   // Asynchronously monitor the execution status of maneuver.
19 Function MonitorSignal(NPC, m):
20 |   if m.execute() = SUCCESS then
21 |   |   NPC.status ← IDLE;
22 |   end
```


generating NPC vehicle behaviors, recording simulation data, and reporting violation scenarios. We illustrate the process of scenario execution and test oracles in DynNPC.

Execution Process. The process of the scenario execution is represented in Algorithm 1. The input of scenario executor includes the map Map, the maximum allowed frame duration t^S, and the configuration of scenario $Conf$. The output of scenario executor is the record of the simulation $Record$.

First, we initialize the scenario (Line 1-5). Specifically, we instantiate the NPC vehicles according to $Conf.\mathbb{N}$ (Line 1). We load the map as well as NPC vehicles into the simulator *sim*, initialize the weather and traffic signals, and set the driving task of the Ego vehicle using $Conf$, bridging it with the ADS under test (Line 2). For each NPC vehicle, we set its status to be $IDLE$ (Line 3-5).

Second, we run the simulation loop for a total of t^S frames (Line 6-16). Each frame represents a simulation time step of 0.1 seconds (Line 15). For each NPC vehicle, if its status is $IDLE$, the NPC vehicle will detect the behavior of the Ego vehicle in the simulator and decide the maneuver m (Line 9), which is introduced in Sec. 3.2. Then, the NPC vehicle will plan the trajectory of m (Line 10), which is introduced in Sec. 3.3, and set its status to $RUNNING$ (Line 11). Besides, we start an asynchronous task to execute and monitor m by calling the function *MonitorSignal* (Line 12). When the simulation is completed, we update the simulation record (Line 17), which includes the trajectories of the Ego vehicle and all the NPC vehicles, and save the record for reproduction and evaluation (Line 18).

For the function *MonitorSignal*, it is an asynchronous function independent of the simulation loop (Line 19-22). We execute the trajectory of maneuver m and wait for the *SUCCESS* signal, which indicates that the NPC vehicle has finished one maneuver (Line 20). Then, we set the status of NPC to $IDLE$ (Line 21) and start to generate another behavior of NPC.

Violation Analysis. The scenario executor will also determine whether there are any violations during the simulation. Based on previous work [15, 32], we consider 4 test oracles to assess the ADS's abilities of collision avoidance, reaching destination, not hitting illegal lines, and obeying traffic signals.

(1) Collision Avoidance. This oracle checks if the Ego vehicle collides with an NPC vehicle. We use the collision detection function provided by the simulator directly to determine the failure of this oracle. Note that, when the collision occurs and is detected by the simulator's callback function, the simulation will end immediately and return the simulation record before the collision.

(2) Not Hitting Illegal Lines. This oracle checks if the Ego vehicle hits the illegal lines (*e.g.,* yellow lines or edge lines) during the simulation. Given the waypoints $\langle p_E^0, p_E^1, \dots, p_E^n \rangle$ of the Ego vehicle and a set of illegal lines denoted as *Lines* extracted from the map, the failing condition of this oracle is defined by Eq. 15,

$$\min(\{D_{E2l}(p_E^t, l) \mid l \in Lines, 0 \leq t \leq n\}) < threshold \tag{15}$$

where $D_{E2l}(p_E^t, l)$ calculates the distance between the center of the Ego vehicle and the illegal line l at frame t, and *threshold* is set to half width of the Ego vehicle bounding box in our work.

(3) Reaching Destination. This oracle checks if the Ego vehicle reaches the destination in the given time. Given the waypoints $\langle p_E^0, p_E^1, \dots, p_E^n \rangle$ of the Ego vehicle after simulation and its destination p_{des}, the failing condition of this oracle is defined by Eq. 16,

$$D_{E2des}(p_E^n, p_{des}) > threshold \tag{16}$$

where $D_{E2des}(p_E^n, p_{des})$ evaluates the final distance of the Ego vehicle to its destination, and *threshold* is set to half length of the Ego vehicle bounding box in our work.

(4) Obeying Traffic Signals. This oracle checks if the Ego vehicle crosses the stop line at a positive speed when the signal is red. Given the waypoints $\langle p_E^0, p_E^1, \dots, p_E^n \rangle$ and the speed sequence $\langle v_E^0, v_E^1, \dots, v_E^n \rangle$ of the Ego vehicle, the traffic signal configuration T_j of the light in Ego direction, and the stop line l_{stop} associated with the traffic light extracted from the map, the failing condition of this oracle is defined by Eq. 17,

$$\exists t \in [0, n], signal(T_j, t) = RED \wedge D_{E2l}(p_E^t, l_{stop}) = 0 \wedge v_E^t > 0 \tag{17}$$

where $signal(T_j, t)$ returns the color of the traffic light T_j at frame t, and $D_{E2l}(p_E^t, l_{stop})$ calculates the distance between the center of the Ego vehicle and the stop line l_{stop} at frame t.

Through the above test oracles, we can discover the violation scenarios. Then, we can collect the simulation record and replay the driving task of the Ego vehicle along with the recorded trajectories of NPC vehicles in the same environment to reproduce the violations for further diagnosis.

4 Evaluation

To evaluate the effectiveness and efficiency of DynNPC, we design the following six research questions.

- **RQ1**: How effective is DynNPC in finding Ego-induced violations compared with state-of-the-art approaches?
- **RQ2**: How efficient is DynNPC in finding Ego-induced violations compared with state-of-the-art approaches?
- **RQ3**: Can DynNPC generate smoother and more various speed sequences of NPC vehicles compared with state-of-the-art approaches?
- **RQ4**: What are the effects of the genetic algorithm-based scenario generator and the three NPC vehicle driving strategies on the testing results?
- **RQ5**: How sensitive is DynNPC to the key safety threshold parameter in maneuver constraints?

- **RQ6**: Can DynNPC be effectively migrated to another simulator in more complex traffic scenarios?

4.1 Evaluation Setup

Target ADS and Simulation Platform. We choose Baidu Apollo 8.0 [7] as our target ADS, which is one of the most representative industrial-grade ADSs with widespread commercialization. We select LGSVL 2021.3 [41] as our simulation platform because LGVSL [62] offers stable connections with Apollo. Although the remote service of LGSVL is no longer maintained, we use a local version [30]. We also evaluate the portability of DynNPC using CARLA 0.9.14 [20].

Prototype. We implement a prototype of DynNPC with 13,657 lines of Python code. Our prototype uses simulator Python APIs [40] for scenario execution and violation detection. During the process of simulation, Apollo 8.0 is equipped with a wide range of sensors, including two camera sensors, one GPS, one radar and one LiDAR. All modules of Apollo are turned on, including localization module, perception module, prediction module, routing module, planning module and control module. We choose the SanFrancisco map in the SVL map library which contains various types of road.

Baselines. We compare DynNPC with four state-of-the-art testing approaches, *i.e.,* AV-Fuzzer [43], AutoFuzz [76], CRISCO [68], and BehAVExplor [15]. AV-Fuzzer uses genetic algorithm to evolve NPC vehicles' maneuvers and speeds after each execution to find violation scenarios in highway and urban-way (*i.e.,* straight and curve roads). AutoFuzz leverages neural networks to predict outcomes and adopts a gradient-based algorithm for the mutation of positions and speeds of traffic participants (*e.g.,* NPC vehicles and pedestrians). CRISCO extracts influential behavior patterns mined from real traffic trajectories (*e.g.,* inD [11]) along with random mutation of traffic participants' speeds to generate testing scenarios after each execution. BehAVExplor is a novel behavior-guided fuzzing approach performing maneuvers mutation and trajectories mutation of NPC vehicles to find diverse violation scenarios. The open-source implementations of these baselines only provide stable integrations with LGSVL. Therefore, we conduct the main comparative experiments on LGSVL to ensure a fair and reproducible comparison.

Research Question Setup. For **RQ1**, **RQ2** and **RQ3**, we run DynNPC and other four approaches for 12 hours, generating driving scenarios respectively. Specifically, we run these approaches in a 4-lane straight road and a crossroad equipped with four traffic lights. Note that, we only run AV-Fuzzer in the four-lane straight road because its open-source version does not support crossroad scenarios and is difficult to migrate. Besides, we set the maximum number of vehicles to 4 in each scenario generated by these approaches, ignoring pedestrians and cyclists, and the maximum simulation duration to 30 seconds for a fair comparison. We verify whether the violations generated by these approaches are induced by the Ego vehicle using DiaVio [48], an LLM-powered diagnosis tool, and ACAV [64], a causality-based analysis tool. These violations are then classified into four groups according to the test oracles they violate (*i.e.,* 4 test oracles in Sec. 3.5).

For **RQ1**, we evaluate the effectiveness of DynNPC from the following aspects: (1) How many EIVs can be found? (2) What is the proportion of EIVs among all the violations? (3) How many unique EIV patterns can be found? To better interpret the discovered EIVs, we group them into *unique EIV patterns*. A unique EIV pattern is defined as a class of Ego-induced violations that share three characteristics: (1) the same violation type exhibited by the Ego vehicle (e.g., rear-end collision, side collision, red-light violation, or destination-missing behavior), (2) the same number of NPC vehicles involved and similar NPC maneuvers triggered, and (3) the same underlying flawed ADS module or causal factor revealed by diagnosis. Two EIVs are assigned to the same pattern only when these three aspects are consistent; otherwise, they are treated as different patterns. The grouping procedure is as follows. We first identify whether a reported violation is induced by the Ego vehicle using DiaVio [48]. For collision violations, we further use ACAV [64] to analyze the causal chain and identify the faulty ADS module. Since ACAV currently only supports collision analysis, the remaining EIVs are further examined through manual

diagnosis based on simulator records, vehicle trajectories, traffic-signal states, and the interaction process between the Ego vehicle and NPC vehicles. After that, we assign each confirmed EIV to a pattern according to the above three criteria. By abstracting a large number of concrete violation scenarios into a small set of recurring and reproducible failure modes, unique EIV patterns provide a more meaningful way to assess whether a testing approach can uncover diverse violation scenarios rather than repeatedly generating homogeneous ones, while also helping developers more efficiently localize the potential flaws in ADSs.

For **RQ2**, we compare DynNPC with other four approaches from two aspects: (1) How much time does it take to find the first EIV and how much time does it take to find one EIV on average? (2) What are the proportions of execution time and analysis time in 12 hours? Specifically, execution time is the time taken to run the scenarios, while analysis time includes processes such as environment initialization, feedback collection, and other computing processes. We also report the average number of NPC maneuver switches triggered in the simulation.

For **RQ3**, we calculate and report the average Standard Deviation of Speed Changes (SDSC) and the number of change points to measure the smoothness and variation of NPC vehicle speed sequences using PELT [36] algorithm.

For **RQ4**, we implement a random approach denoted as Rand, which performs online NPC behavior generation but randomly generates scenario initial configurations. Besides, we create three variants of DynNPC, *i.e.,* DynNPC-Yield, DynNPC-Adver and DynNPC-Over, which generate scenarios where all NPC vehicles would only adopt yielding, adversarial, or overtaking driving strategy throughout the simulation, respectively. We run these variants and evaluate them from the aspects concerned in **RQ1**.

For **RQ5**, we study the sensitivity of DynNPC to the key safety threshold parameter in maneuver constraints regarding acceleration, deceleration, and lane changing maneuvers. By varying the threshold parameter from 20 meters to 40 meters, we analyze whether the effectiveness and efficiency of DynNPC remain stable under different settings.

For **RQ6**, we migrate DynNPC to CARLA 0.9.14 [20] using the official map named Town03, which is a larger, urban map with a roundabout and large junctions. We set the maximum number of vehicles to 8 in each scenario generated by DynNPC, and the maximum simulation duration to 30 seconds. We report the average results of the effectiveness and efficiency of DynNPC in the new environment.

We run all the above experiments 5 times in case of randomness, applying Mann-Whitney-U-Test [52] and computing the Cohen's d [16] to evaluate the statistical significance and effect size. We replay the violation scenarios to ensure the reproductivity of the failures, and finally report the average results.

Experiment Environment. We conduct all the experiments on an Ubuntu 22.04.4 LTS server with an NVIDIA GeForce RTX 4090 GPU, Intel Core i9-13900K (32) CPU with 5.500GHz processor and 64GB memory.

4.2 RQ1: Effectiveness Evaluation

We first present the overall effectiveness of DynNPC and then discuss the unique EIV patterns we found in detail.

Overall Results. For each 12-hour run, on average, 770.6 scenarios and 643.0 scenarios are generated by DynNPC on the straight road and the crossroad, respectively. After manual verification, Table 1 presents the general effectiveness of DynNPC in finding EIVs compared with other approaches. In terms of the number of EIVs, DynNPC surpasses the other four approaches by 102.37% at least on the straight road, and by 47.26% at least on the crossroad. In terms of the proportion of EIVs among all reported violations, DynNPC achieves the best performance at 80.65% (153.4/190.2) in the straight road scenarios, whereas AV-Fuzzer yields the lowest proportion at only 14.89% (36.6/245.8). For the crossroad scenarios, DynNPC outperforms other approaches, achieving an average of 82.12% (69.8/85.0), while AutoFuzz performs the worst with only 13.52% (15.2/112.4). Overall, the proportion of the violations induced by the Ego vehicle increases by 125.21% on average compared

Table 1. Results of the General Effectiveness

Road	Approach	#Scenario[1]	#Violation	#EIV[2]	Proportion	#UniP[3]
Straight Road	AV-Fuzzer	665.8	245.8	36.6	14.89%	4.8
	AutoFuzz	893.4	160.6	27.6	17.19%	3.0
	CRISCO	913.6	138.4	67.4	48.70%	6.6
	BehAVExplor	645.4	101.8	75.8	74.46%	10.2
	DynNPC	770.6	190.2	**153.4**	80.65%	**11.8**
Crossroad	AutoFuzz	944.8	112.4	15.2	13.52%	2.6
	CRISCO	973.2	104.0	33.4	32.12%	3.2
	BehAVExplor	611.8	84.8	47.4	55.90%	4.4
	DynNPC	643.0	85.0	**69.8**	**82.12%**	**7.2**

[1] *the number of the generated scenarios.*
[2] *the number of violations induced by the Ego vehicle.*
[3] *the number of unique EIV patterns.*

with other approaches. In terms of the number of unique EIV patterns, DynNPC identifies 11.8 and 7.2 patterns on the straight road and crossroad, respectively, outperforming the best results of other approaches, which achieve 10.2 and 4.4 (both by BehAVExplor). DynNPC improves the number of discovered unique EIV patterns, on average, by at least 39.66%.

Breakdown Analysis. We further analyze why the baseline approaches exhibit much higher false-positive rates than DynNPC, and why DynNPC still cannot eliminate false positives completely. The higher false-positive rates of the baseline approaches mainly stem from their pre-execution generation of NPC behaviors. AV-Fuzzer mutates predefined maneuvers and speeds arbitrarily, which can easily produce overly aggressive interactions. AutoFuzz improves search efficiency neural-guided mutation without ensuring behavior-level reasonableness at runtime, leading to many NPC vehicles appearing in unreasonable locations. CRISCO leverages behavior patterns mined from real trajectories, yet these patterns are still instantiated before execution and cannot adapt to the Ego vehicle's real-time responses. BehAVExplor emphasizes behavior diversity during exploration, but does not explicitly constrain whether NPC decisions remain reasonable under instantaneous signal. As a result, violations reported by these baselines are more likely to be dominated by unreasonable NPC behaviors, leading to higher false-positive rates. The most common violation scenarios generated by these baseline approaches with unreasonable NPC behaviors are shown in Fig. 1.

In contrast, DynNPC reduces such false positives by generating NPC maneuvers and trajectories online according to the Ego vehicle's runtime behavior and traffic signals, thereby making NPC behaviors more interaction-aware and behaviorally constrained. Nevertheless, DynNPC still cannot completely eliminate false positives, because its rationality guarantees are implemented through a finite set of maneuver constraints with safety thresholds rather than a complete liability model of realistic driving. Consequently, some borderline cases may still satisfy the designed constraints while remaining overly harsh or effectively unavoidable for the Ego vehicle, especially when the NPC vehicles need to perform lane changing maneuvers. The current maneuver constraints use a safety threshold of 30 meters by default. Although this threshold filters out many clearly unreasonable cut-ins, under the aggressive driving strategy, the speed profile generated by the *s-t* graph may still produce borderline interactions in which the NPC's lane change leaves the Ego vehicle with insufficient time to avoid a rear-end collision. Such cases satisfy the implemented constraints but may still be practically unavoidable for the Ego vehicle, and are therefore counted as false positives. As further discussed in **RQ5**, increasing the safety threshold can reduce such false positives by making NPC behaviors more conservative.

Table 2. Results of the Number of Unique EIV Patterns

Approach	#R1[1]	#R2[2]	#R3[3]	#R4[4]	Sum	Details
AV-Fuzzer	5	0	0	0	5	R1-1~R1-5
AutoFuzz	4	0	2	0	6	R1-1, R1-2, R1-8, R1-9, R3-3, R3-4
CRISCO	7	2	2	0	11	R1-1~R1-3, R1-6, R1-8~R1-10, R2-1, R2-3, R3-3, R3-4
BehAVExplor	9	3	4	0	16	R-1~R1-5, R1-7, R1-8~R1-10, R2-1~R2-3, R3-1~R3-4
DynNPC	**12**	**3**	**4**	**1**	**20**	R1-1~R1-12, R2-1~R2-3, R3-1~R3-4, R4-1

[1] *the number of patterns where the Ego vehicle collides with NPC vehicles.*
[2] *the number of patterns where the Ego vehicle hits illegal lines.*
[3] *the number of patterns where the Ego vehicle gets stuck that fails to reach the destination.*
[4] *the number of patterns where the Ego vehicle runs red lights.*

To sum up, DynNPC effectively identifies more EIVs in a given time. This success is attributed to its dynamic generation of NPC vehicle maneuvers and trajectories during each simulation execution. By regulating the rationality of NPC behaviors, DynNPC minimizes unreasonable behaviors of NPC vehicles, thereby reducing the false positive rate of reported violations. Moreover, DynNPC introduces mutations to NPC driving strategies, ensuring behavioral diversity throughout the simulation. In contrast, other approaches overly pursue finding more violations (*e.g.,* collision scenarios or hitting illegal line scenarios) and ignore the rationality of NPC vehicle behaviors (*e.g.,* abrupt speed changes or non-compliance with traffic signals), resulting in a high false positive rate of reported violations.

Unique EIV Patterns. Table 2 lists the total number of unique EIV patterns that violate different test oracles in all 5 repetitions of experiments in these 2 types of road (*i.e.,* straight road and crossroad). DynNPC finds 20 unique EIV patterns in total with 12 patterns where the Ego vehicle collides with NPC vehicles, 3 patterns where the Ego vehicle hits illegal lines, 4 patterns where the Ego vehicle gets stuck and fails to reach the destination and 1 pattern where the Ego vehicle runs red lights. The result shows that DynNPC can discover the most number of unique EIV patterns and cover all unique EIV patterns found by baselines. Fig. 9 shows an overview of each EIV pattern, where the red car denotes the Ego vehicle and green cars represent NPC vehicles. We provide an in-depth discussion for each of these unique EIV patterns as follows.

Case Study R1-1. When the NPC vehicle initiates a lane change, it satisfies all behavior constraints, including maintaining a safe longitudinal distance, activating the turn signal, and performing the maneuver in a non-solid-line area. Finally, the NPC vehicle finishes the lane change, however, the Ego vehicle fails to decelerate in time, leading to a collision. This case shows a potential issue in the prediction module, as the Ego vehicle has not accurately anticipated the speed of NPC vehicle's lane changing behavior.

Case Study R1-2. When the Ego vehicle changes lanes, it fails to maintain a safe longitudinal distance and avoid the NPC vehicle approaching from behind in the target lane, resulting in a collision. According to the root cause analysis, this case shows potential issues in the prediction and planning module, as the Ego vehicle have miscalculated the speed and position of the NPC vehicle and give a wrong driving decision.

Case Study R1-3. The NPC vehicle performs a compliant lane change with a low speed, activating the turn signal and keeping a sufficient distance before merging. The Ego vehicle chooses to accelerate forward, but ignores another NPC vehicle that is decelerating about 30 meters ahead, ultimately leading to a collision. This case shows that the Ego vehicle lacks effective response to lane changing behavior and risk assessment, with flaw in the planning module.

Case Study R1-4. Two NPC vehicles collide during a lane change and come to a stop ahead. After the collision, both NPCs stop far from the Ego vehicle and activate their brake lights. According to the root cause analysis, the Ego vehicle incorrectly perceives the two stationary vehicles as a single large obstacle in the adjacent lane, while losing the perception of the NPC vehicle blocking its current lane, and fails to stop in time. This suggests that the

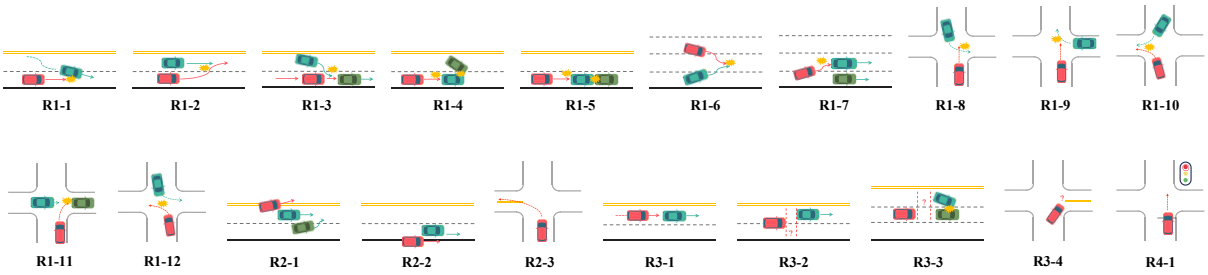


Fig. 9. Overview of Each Ego-induced Violation Pattern

Ego vehicle has limitations in the perception module, which is responsible for object detection and classification errors.

Case Study R1-5. The Ego vehicle is following two NPC vehicles. When the two NPC vehicles collide and stop abruptly, the Ego vehicle fails to react in time because the bounding boxes of the two stopped vehicles overlap in the perception results, making it difficult to accurately estimate the safe distance to the obstacles ahead, resulting in a collision with the NPC vehicles. This case suggests a lack of effective obstacle perception and safe distance estimation.

Case Study R1-6. The Ego vehicle and an NPC vehicle attempt to change into the same lane simultaneously. Due to the weak lateral awareness and poor coordination, the Ego vehicle fails to detect the NPC vehicle's movement according to root cause analysis, resulting in a side collision during the merging process.

Case Study R1-7. The Ego vehicle is following a slow-moving NPC vehicle and decides to change lanes to overtake. However, the Ego vehicle fails to avoid another NPC vehicle in the adjacent lane and collides with it during the lane change, indicating problems in the perception and planning modules according to root cause analysis.

Case Study R1-8. The Ego vehicle is going straight through an intersection when an NPC vehicle from the opposite direction makes a left turn. Since the NPC vehicle is moving slowly, the Ego vehicle marks it with a *Ignored* tag by wrong priority prediction and fails to properly predict its turning behavior, resulting in a side collision.

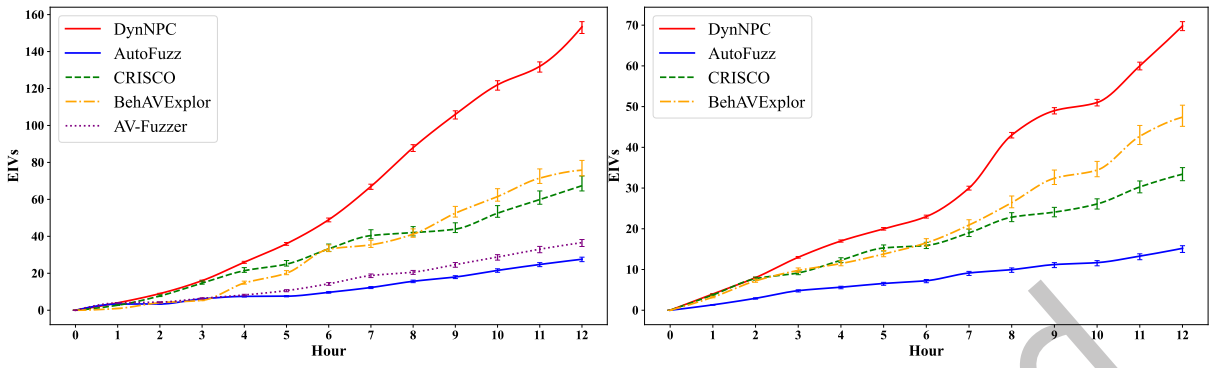
Case Study R1-9. The Ego vehicle is going straight through an intersection when an NPC vehicle from the opposite direction makes a right turn across its path. Due to the slow speed of the NPC vehicle, the Ego vehicle tags it as *Ignored* by wrong priority prediction and does not yield or decelerate, colliding on the side of the NPC vehicle.

Case Study R1-10. The Ego vehicle is making a left turn at an intersection while an NPC vehicle from the opposite direction is making a right turn. The NPC vehicle is moving slowly, leading the Ego vehicle to assign it an *Ignored* tag by wrong priority prediction and proceed without sufficient caution, resulting in a side collision.

Case Study R1-11. When the Ego vehicle turns right to merge into traffic, it predicts the speed of the slowly moving vehicle behind a fast vehicle wrongly, and assigns the slow vehicle an *Ignored* tag. Consequently, the Ego vehicle collides with the NPC vehicle while turning, showing a problem in the prediction module.

Case Study R1-12. The Ego vehicle and an NPC vehicle simultaneously turn left from opposite directions at an intersection. Due to inaccurate prediction of the NPC vehicle's trajectory and the shaking of the Ego vehicle when turning, a side collision occurs. Root cause analysis indicates that there are problems in the prediction module and the control module when handling intersection turning behavior.

Case Study R2-1. The Ego vehicle changes lanes across the yellow line when overtaking if the NPC vehicles block the feasible lanes, indicating a problem in the planning module.



(a) Straight Road (b) Crossroad

Fig. 10. Results of the General Efficiency

Case Study R2-2. The Ego vehicle moves on the road boundaries to perform the overtaking behavior, indicating a problem in the planning module.

Case Study R2-3. In a left-turn scenario, if the speed of the Ego vehicle is slow and the steering angle is relatively large, the Ego vehicle will hit the yellow line and continue to press the yellow line after left turn, indicating a problem in the control module.

Case Study R3-1. The Ego vehicle follows a slow-moving NPC vehicle without an overtaking maneuver, despite having sufficient conditions to do so. As a result, the Ego vehicle fails to reach the destination within the required time limit. This indicates a problem in the planning module regarding overtaking strategies in low-speed following scenarios.

Case Study R3-2. The Ego vehicle follows an NPC vehicle whose speed is slow, and it hesitates between continuing to follow and initiating a lane change maneuver. As a result, it becomes stuck in decision-making and fails to complete an overtaking. This case reveals limitations in the planning module.

Case Study R3-3. Two NPC vehicles collide ahead of the Ego vehicle, blocking the lane and forming a static obstacle. The Ego vehicle fails to plan a feasible path to change lanes and bypass the obstacle, resulting in it being stuck behind the blocked lane. This indicates limitations in the routing and planning modules.

Case Study R3-4. When the turning arc is small, the Ego vehicle may plan off-road trajectories, leading to fluctuating steering behavior. Eventually, the vehicle gets stuck, showing issues in both the planning and control modules.

Case Study R4-1. The Ego vehicle stops when the signal is red and passes the stop line. Then, it loses the perception of the traffic signal and restarts to run a red light through the intersection, indicating a bug in the perception module.

Summary. DynNPC can not only find more Ego-induced violations compared with other approaches, increasing the proportion of EIVs among all reported violations, on average, by 125.21%, but also can discover more unique EIV patterns effectively, improving the number of discovered unique EIV patterns, on average, by at least 39.71%.

4.3 RQ2: Efficiency Evaluation

Fig. 10 and Table 3 presents the efficiency of DynNPC in finding EIVs. For the straight road scenarios, DynNPC achieves the shortest time to identify the first EIV, requiring only 4.69 minutes, with CRISCO ranking second. In

Table 3. Efficiency in Finding Ego-induced Violations

Road Type	Approach	First EIV[1]	One EIV[2]
Straight Road	AV-Fuzzer	80.70	19.31
	AutoFuzz	130.15	25.71
	CRISCO	7.40	10.69
	BehAVExplor	31.83	9.44
	DynNPC	**4.69**	**4.67**
Crossroad	AutoFuzz	107.87	44.17
	CRISCO	**10.69**	21.36
	BehAVExplor	29.43	15.22
	DynNPC	15.30	**10.14**

[1] *the time used to find the first EIV (minute).*
[2] *the average time used to find one EIV (minute).*

Table 4. Results of Performance Characteristics

Time	AV-Fuzzer	AutoFuzz	CRISCO	BehAVExplor	DynNPC
Execution	10.65	11.25	11.30	10.82	11.18
Analysis	1.35	0.75	0.70	1.18	0.82

contrast, AutoFuzz exhibits the poorest performance, taking 130.15 minutes. For the crossroad scenarios, CRISCO takes the shortest time (10.69 minutes) and AutoFuzz takes the longest time (107.87 minutes) to find the first EIV, while DynNPC uses 15.30 minutes. Overall, DynNPC improves efficiency in finding the first EIV by 92.50% on the straight road, and by 68.98% on the crossroad, on average, compared with other approaches.

In terms of the average time to find an EIV, DynNPC consistently ranks first across both straight road and crossroad scenarios among all the approaches, achieving an average time of 7.41 minutes. Notably, although CRISCO identifies EIVs quickly in the early stages by leveraging trajectories from real-world datasets, its later use of random trajectory mutations introduces numerous unreasonable NPC behaviors, resulting in less efficient overall performance. On average, DynNPC achieves a reduction in the time to find one EIV ranging from 41.96% to 82.94%. Results show that the dynamic generation of NPC vehicle behaviors during simulation execution can significantly accelerate the search for EIVs.

In addition, we calculate the average execution and analysis time across all scenarios for each approach. Table 4 shows the results in detail. DynNPC uses, on average, 0.82 hours for analysis and 11.18 hours for execution during the simulation. AutoFuzz and CRISCO have shorter analysis time than DynNPC at the cost of the diversity of NPC vehicle trajectories. Specifically, AutoFuzz employs a constrained neural network evolutionary search method to generate scenarios where NPC vehicles seldom change speeds, thereby reducing the complexity of NPC vehicle trajectory computation. CRISCO, on the other hand, constructs NPC vehicle waypoints by randomly mutating the combination of predefined influential behavior patterns extracted from datasets, leading to reduced computational overhead. Compared with AV-Fuzzer and BehAVExplor, DynNPC reduces the analysis time by 39.26% and 30.51%, respectively. The analysis time of AV-Fuzzer and BehAVExplor are prolonged due to complex computation of NPC vehicle trajectories before each execution. However, DynNPC puts this task into parallel during the execution process, reducing the additional analysis time. In general, DynNPC reduces the analysis time by 39.71% on average than other approaches.

To further characterize how dynamically interactive the generated NPC behaviors are in practice, we additionally measure the average number of NPC vehicle maneuver switches triggered during simulation. The results

Table 5. Smoothness and Variation of Speed Sequences

Road Type	Trajectory	SDSC[1]	#ChangeP[2]
Straight Road	**AV-Fuzzer**	2.08	24.42
	AutoFuzz	0.18	0.43
	CRISCO	0.01	0.00
	BehAVExplor	0.33	3.26
	DynNPC	0.12	2.81
Crossroad	**AutoFuzz**	0.15	1.89
	CRISCO	0.01	0.00
	BehAVExplor	0.35	6.20
	DynNPC	0.09	5.70

[1] *the standard deviation of speed changes.*
[2] *the number of change points in speed sequences.*

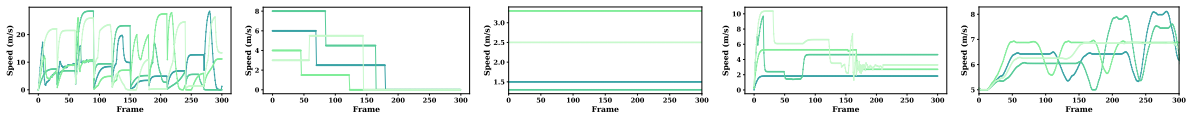
show a clear difference between DynNPC and most baselines. In AutoFuzz, CRISCO, and BehAVExplor, each NPC vehicle follows a predefined behavior once the scenario starts, and thus the average number of maneuver switches during execution is 0. AV-Fuzzer is the only exception, as its NPC vehicles randomly switch maneuvers (*i.e.,* going straight and changing lane) every five seconds, resulting in an average of 3.76 switches on the straight road. In contrast, DynNPC enables NPC vehicles to adjust their behaviors online according to the Ego vehicle's real-time states and traffic conditions, leading to an average of 3.25 maneuver switches per NPC on the straight road and 1.12 on the crossroad. These results indicate that the NPC vehicles generated by DynNPC are not limited to executing a single predefined maneuver, but instead exhibit substantial runtime behavioral adaptation. This dynamic interaction is especially frequent on the straight road, where repeated acceleration, deceleration, lane changing and parking decisions can be triggered during a single run, while it is relatively lower at crossroads due to shorter interaction windows and stronger traffic-signal constraints. Such runtime maneuver adaptation also helps improve testing efficiency, as it allows NPC vehicles to produce richer and more targeted interactions with the Ego vehicle within a single run, thereby increasing the chance of exposing EIVs without relying on repeated exploration of many pre-defined trajectory variants.

Summary. DynNPC can find Ego-induced violations more quickly compared with other approaches, reducing the time to find the first EIV and the average time to find one EIV by 82.13% and 60.70%, respectively. Besides, DynNPC maintains competitive runtime overhead while enabling dynamically interactive NPC behaviors, which helps improve testing efficiency by producing richer and more targeted interactions within a single run.

4.4 RQ3: Smoothness and Variation of NPC Vehicle Speeds

Table 5 presents the smoothness and variation of NPC vehicle speed sequences generated by different approaches within the simulator. AV-Fuzzer has the highest standard deviation of speed changes (SDSC) and the greatest number of change points (#ChangeP) in straight road scenarios, indicating that the speed sequences of NPC vehicles generated by AV-Fuzzer exhibit excessive fluctuations and lack smoothness. This is because AV-Fuzzer divides the driving scenario into 5-second time slices and incorporates the behavior of NPC vehicles in each time slice into its genetic representation. During the mutation process, it alters the speed of vehicles within each time slice, leading to significant and abrupt speed changes illustrated by an example of speed sequences shown in Fig. 11a.

AutoFuzz demonstrates low SDSC values both on the straight road and crossroad. However, its number of change points is notably low (0.43 on straight roads and 1.89 on crossroads), indicating that it generates overly



(a) AV-Fuzzer (b) AutoFuzz (c) CRISCO (d) BehAVExplor (e) DynNPC

Fig. 11. Examples of Speed Sequences Generated by Different Approaches.

simplistic speed profiles with minimal variations. As shown in Fig. 11b, AutoFuzz adjusts the fixed speed of vehicles over specific time periods in the scenario configurations based on test feedback, ultimately bringing the vehicle to a stop on the map. It fails to avoid occasional abrupt speed transitions and its lack of dynamic changes in speed sequences may fail to capture the complexity of real-world driving behavior.

CRISCO produces nearly constant speed sequences of NPC vehicles, with SDSC values close to zero and no change points in both straight road scenarios and crossroad scenarios. As shown in Fig. 11c, the NPC vehicles in CRISCO drive with fixed speeds specified in the scenario configurations. While this approach ensures the maximum smoothness, it fails to simulate diverse driving behaviors, as real-world driving inevitably involves various speed variations.

BehAVExplor performs best among the four baselines, generating NPC vehicle speed sequences with acceptable SDSC values and #ChangeP values in both straight road scenarios and crossroad scenarios. It constrains the speed of each maneuver taken by NPCs. However, as shown in Fig. 11d, it still inevitably introduces unreasonable speed changes of NPC vehicles in the scenario configurations due to the guidance of finding collisions as much as possible.

With respect to DynNPC, on the straight road, it exhibits an SDSC of 0.12 and a #ChangeP of 2.81, on average. On the crossroad, it achieves an SDSC of 0.09 and a #ChangeP of 5.70, on average, indicating that our generated speed profiles are not simple constant-speed lines but contain meaningful variations. As an example of the speed sequences generated by DynNPC, shown in Fig. 11e, the speed sequences are smooth and have moderate variations, indicating that DynNPC can generate speed sequences with better smoothness and speed variations compared with other four approaches.

Summary. DynNPC can generate speed sequences with better smoothness and variations compared with other four approaches. The planning of speed during simulation execution based on *s-t* graph ensures the smoothness and the mutation of driving strategy improves the variation of speed sequences.

4.5 RQ4: Effects of GA-based Generator and Different Driving Strategies

We first evaluate the effect of the GA-based scenario generator by comparing DynNPC with Rand, and then analyze the effects of different driving strategies using the three variants of DynNPC.

Table 6 presents the effects of the GA-based scenario generator on the general effectiveness. Compared with Rand, DynNPC consistently finds more EIVs and more unique EIV patterns in both straight road and crossroad scenarios. Specifically, on the straight road, DynNPC increases the average number of EIVs from 88.7 to 153.4 and the number of unique EIV patterns from 5.8 to 11.8. On the crossroad, DynNPC improves the average number of EIVs from 53.9 to 69.8 and the number of unique EIV patterns from 2.6 to 7.2. The proportion of EIVs generated by DynNPC is also slightly higher than that of Rand in both scenarios.

Although Rand generates more scenarios within 12 hours, its effectiveness is clearly worse than DynNPC considering the violation rate. Besides, during five repeated experiments, Rand only find 9 unique EIV patterns (*i.e.,* R1-1, R1-2, R1-3, R1-8, R1-9, R2-2, R3-1, R3-3, and R3-4) in 12 hours. These results indicate that the initial

Table 6. Effects of GA-based Generator on the General Effectiveness

Road	Approach	#Scenario[1]	#Violation	#EIV[2]	Proportion	#UniP[3]
Straight Road	RAND	910.8	112.0	88.7	79.20%	5.8
	DYNNPC	770.6	190.2	**153.4**	**80.65%**	**11.8**
Crossroad	RAND	794.8	66.4	53.9	80.11%	2.6
	DYNNPC	643.0	85.0	**69.8**	**82.12%**	**7.2**

[1] *the number of the generated scenarios.*
[2] *the number of violations induced by the Ego vehicle.*
[3] *the number of unique EIV patterns.*

Table 7. Effects of Different Driving Strategies on the General Effectiveness

Road	Approach	#Scenario[1]	#Violation	#EIV[2]	Proportion	#UniP[3]
Straight Road	DYNNPC-YIELD	767.4	228.8	185.4	**81.03%**	7.8
	DYNNPC-ADVER	792.8	424.4	**291.0**	68.57%	10.0
	DYNNPC-OVER	734.0	161.0	38.8	24.10%	4.4
	DYNNPC	770.6	190.2	153.4	80.65%	**11.8**
Crossroad	DYNNPC-YIELD	626.0	68.8	58.4	**84.88%**	4.8
	DYNNPC-ADVER	672.4	161.0	**97.0**	60.25%	5.0
	DYNNPC-OVER	651.4	42.4	7.0	16.51%	1.8
	DYNNPC	643.0	85.0	69.8	82.12%	**7.2**

[1] *the number of the generated scenarios.*
[2] *the number of violations induced by the Ego vehicle.*
[3] *the number of unique EIV patterns.*

scenario configurations, such as the initial positions of NPC vehicles, traffic signal settings, and environment conditions, still largely determine whether the Ego vehicle and NPC vehicles can enter meaningful interactions. By adopting genetic algorithm to optimize these high-level scenario configurations, DynNPC is able to generate fewer but more effective scenarios, thereby exposing substantially more EIVs and diverse EIV patterns. Therefore, the GA-based generator is complementary to runtime NPC behavior generation, rather than redundant with it.

Then, we run the three variants of DynNPC (*i.e.,* DynNPC-Yield, DynNPC-Adver and DynNPC-Over) and present the effects of different driving strategies on the general effectiveness in Table 7.

DynNPC-Yield ranks first in terms of the proportion of EIVs in the straight road and crossroad scenarios, respectively. In the scenarios generated by DynNPC-Yield, NPC vehicles typically travel at lower speeds than the Ego vehicle. However, Apollo exhibits poor performance in predicting slow-moving objects, and its overly conservative following strategy leads to failure in reaching the destination when traveling behind a low-speed NPC vehicle.

DynNPC-Adver generates the most EIVs on both straight road and crossroad. This is because the adversarial driving strategy adopted by NPC vehicles effectively increases interactions with the Ego vehicle, raising the difficulty for the Ego vehicle to complete its driving tasks. However, this strategy also leads to more scenarios where the NPC vehicle aggressively collides with a decelerating Ego vehicle, resulting in a higher false positive rate than DynNPC-Yield.

DynNPC-Over generates the fewest violations. Due to the high speeds of NPC vehicles in its scenarios, most of the NPC vehicles hit the stopped Ego vehicle from behind or from the side aggressively in the reported violations. As a result, the proportion of EIVs found by DynNPC-Over accounts for only 24.10% and 16.51% on the straight road and crossroad, respectively.

Table 8. Effects on the Number of Unique EIV Patterns

Method	#R1[1]	#R2[2]	#R3[3]	#R4[4]	Sum	Details
DynNPC-Yield	7	3	3	1	14	R1-1, R1-4~R1-9, R2-1~R2-3, R3-1, R3-3, R3-4, R4-1
DynNPC-Adver	9	3	3	1	16	R1-1~R1-7, R1-10, R1-12, R2-1~R2-3, R3-2~R3-4, R4-1
DynNPC-Over	4	1	1	1	7	R1-4~R1-7, R2-3, R3-3, R4-1
DynNPC	**12**	**3**	**4**	**1**	**20**	R1-1~R1-12, R2-1~R2-3, R3-1~R3-3, R4-1

[1] *the number of patterns where the Ego vehicle collides with NPC vehicles.*
[2] *the number of patterns where the Ego vehicle hits illegal lines.*
[3] *the number of patterns where the Ego vehicle gets stuck that fails to reach the destination.*
[4] *the number of patterns where the Ego vehicle runs red lights.*

Table 9. Sensitivity of the Key Safety Threshold Parameter

Threshold	#Scenario[1]	#Violation	#EIV[2]	Proportion	#Unip[3]
DynNPC-20	772.2	404.8	141.9	35.05%	7.0
DynNPC-30	706.8	137.6	116.6	81.39%	9.5
DynNPC-40	512.4	82.8	78.6	94.93%	5.5

[1] *the number of the generated scenarios.*
[2] *the number of violations induced by the Ego vehicle.*
[3] *the number of unique EIV patterns.*

In terms of the number of unique EIV patterns, all these three variants can only find part of the patterns discovered by DynNPC in 12 hours. DynNPC-Adver performs best among these three variants. DynNPC-Over finds the fewest unique EIV patterns with only an average of 4.4 ones on the straight road and 1.8 ones on the crossroad. Table 8 shows the total unique patterns of each variant in 5 repetitions of experiments. The result shows that one single driving strategy in scenarios is not conducive to improving the diversity of violation scenarios found.

Summary. The GA-based scenario generator and runtime NPC behavior generation are complementary in DynNPC. The former improves the effectiveness of scenario initialization, while the latter increases interaction richness during execution. Besides, NPC vehicles employing yielding and adversarial strategies pose greater threats to the Ego vehicle, while the mutation of driving strategies further contributes to the diversity of unique EIV patterns.

4.6 RQ5: Sensitivity Analysis of Key Safety Threshold Parameters

Table 9 presents the sensitivity analysis results of DynNPC under different values of the key safety threshold parameter (denoted as DynNPC-20, DynNPC-30, and DynNPC-40, respectively). The reported results are averaged over the straight road and crossroad scenarios, where each scenario type is repeated 5 times.

When the threshold is set to 20 meters, DynNPC finds the largest number of EIVs. Increasing the threshold from 20 meters to 30 meters reduces the average number of EIVs by 17.83%, and further increasing it to 40 meters leads to a total reduction of 44.61%. This is because a smaller safety threshold allows NPC vehicles to trigger acceleration, deceleration, and lane-changing maneuvers at a shorter distance from the Ego vehicle, leaving the ADS less reaction time and thus creating more opportunities for violations.

However, such close-range interactions also introduce substantially more false positives. Compared with DynNPC-20, the proportion of EIVs among all reported violations increases by 132.21% under DynNPC-30 and by 170.84% under DynNPC-40. Equivalently, the false-positive rate decreases from 64.95% under DynNPC-20 to 18.61% under DynNPC-30 and 5.07% under DynNPC-40. This indicates that when the threshold becomes larger, NPC vehicles switch strategies farther away from the Ego vehicle, leaving the ADS more reaction distance and

Table 10. Unique EIV Patterns under Different Safety Thresholds

Method	#R1[1]	#R2[2]	#R3[3]	#R4[4]	Sum	Details
DynNPC-20	8	3	2	1	14	R1-2, R1-4~R1-5, R1-8~R1-12, R2-1~R2-3, R3-3~R3-4, R4-1
DynNPC-40	6	2	2	1	11	R1-4~R1-5, R1-8~R1-10, R1-12, R2-1, R2-3, R3-3~R3-4, R4-1
DynNPC-30	**12**	**3**	**4**	**1**	**20**	R1-1~R1-12, R2-1~R2-3, R3-1~R3-3, R4-1

[1] *the number of patterns where the Ego vehicle collides with NPC vehicles.*
[2] *the number of patterns where the Ego vehicle hits illegal lines.*
[3] *the number of patterns where the Ego vehicle gets stuck that fails to reach the destination.*
[4] *the number of patterns where the Ego vehicle runs red lights.*

Table 11. Effectiveness and Efficiency of DynNPC in Roundabout Scenarios using CARLA

	#Scenario[1]	#Violation	#EIV[2]	Proportion	#Unip[3]	First EIV[4]	One EIV[5]
DynNPC	495.2	386.8	317.4	82.06%	9.2	2.24	2.26

[1] *the number of the generated scenarios.*
[2] *the number of violations induced by the Ego vehicle.*
[3] *the number of unique EIV patterns.*
[4] *the time used to find the first EIV (minute).*
[5] *the average time used to find one EIV (minute).*

correspondingly reducing overly aggressive or unreasonable interactions. As a result, the reported violations are much more likely to be EIVs.

In terms of the number of unique EIV patterns, DynNPC-30 performs the best, which is 35.71% higher than DynNPC-20 and 72.73% higher than DynNPC-40. This suggests that an excessively small threshold may introduce too many aggressive interactions, many of which repeatedly trigger similar EIVs, while an overly large threshold makes the generated scenarios less challenging and reduces behavioral diversity. Therefore, a moderate threshold can better balance interaction intensity and behavioral reasonableness, thus helping expose more diverse EIV patterns. Table 10 presents the details of the unique EIV patterns found by DynNPC under different thresholds.

Overall, the key safety threshold parameter has a clear impact on the effectiveness of DynNPC. A smaller threshold helps find more violations, but also introduces substantially more false positives. A larger threshold improves the proportion of EIVs by giving the ADS more reaction distance, but it also reduces the number of discovered violations and weakens the interaction intensity. Among the three settings, 30 meters provides the best trade-off, as it maintains a relatively high EIV proportion, while achieving the highest diversity of unique EIV patterns.

Summary. The key safety threshold parameter has a clear impact on the effectiveness of DynNPC. A smaller threshold helps increase interaction density, while a larger threshold improves the EIV proportion but reduces both the number and diversity of discovered EIVs. In our evaluation, the setting of 30 meters provides the best trade-off.

4.7 RQ6: Portability Evaluation

Table 11 presents the effectiveness and efficiency of DynNPC after migrating it to CARLA in roundabout scenarios.

The results show that DynNPC remains effective in the new simulator and more complex traffic environment. Within 12 hours, DynNPC generates 495.2 scenarios and reports 386.8 violations, among which 317.4 are EIVs, yielding an EIV proportion of 82.06%. Besides, DynNPC discovers 9.2 unique EIV patterns on average in this setting. We further analyze the diversity of discovered EIVs in the roundabout scenarios. Although the roundabout we used is different from a standard signalized intersection, its two-lane structure and multiple junction areas still create conflict points similar to those in straight road and intersection scenarios, such as merging, yielding and turning conflicts. Across 5 repetitions, DynNPC reveals 11 unique EIV patterns, including R1-1, R1-2, R1-3,

R1-4, R1-5, R1-9, R1-11, R3-1, R3-2, R3-3, and R3-4. These results indicate that the core mechanism of DynNPC, including runtime NPC maneuver decision and online trajectory generation, can still effectively expose ADS weaknesses after being migrated from LGSVL to CARLA.

In terms of efficiency, DynNPC finds the first EIV in 2.24 minutes and requires 2.26 minutes on average to find one EIV. Moreover, the average analysis time and execution time are 0.84 hours and 11.16 hours, respectively. Although the maximum number of NPC vehicles is increased from 4 in LGSVL-based experiments to 8 in CARLA, the additional computational overhead remains limited. This is because DynNPC performs behavior generation and monitoring asynchronously during simulation execution, which amortizes the computation cost and avoids introducing substantial extra analysis overhead before each run. These results suggest that the dynamic maneuver decision logic and asynchronous execution design of DynNPC can scale to more complex traffic environments without significantly sacrificing efficiency.

Overall, these results demonstrate the portability of DynNPC across simulators and scenario types. Although our main comparison with prior work is conducted on LGSVL for fairness, DynNPC can be effectively migrated to CARLA and remains capable of finding EIVs effectively and efficiently in complex scenarios.

Summary. DynNPC can be effectively migrated to another simulator and remains both effective and efficient in more complex roundabout scenarios with more NPC vehicles, demonstrating its portability.

5 Threats to Validity

First, the selection of target ADS and simulator poses a threat to validity. We select Apollo 8.0 as our target ADS, which is an open-source ADS and widely used in the industry, to ensure fair comparisons with baselines that are also only adapted to this version. We choose LGSVL rather than CARLA because it has good compatibility with Apollo 8.0 and all the baselines support LGSVL. Newer versions of Apollo are not considered due to unstable bridging with simulators. In addition, we migrate DynNPC to CARLA to demonstrate its portability.

Second, the selection of baselines poses another threat to validity. To mitigate this threat, we select four state-of-the-art approaches that support Apollo and LGSVL. AV-Fuzzer and BehAVExplor are based on a fuzzing engine using genetic algorithm. AutoFuzz is one of the newest testing approaches guided by neural network and CRISCO is a data-driven approach using influential behavior patterns derived from real-world dataset. It should be noted that the traffic participants in DynNPC currently support NPC vehicles, while AutoFuzz and CRISCO support pedestrians and cyclists. During the experiments, all approaches only use vehicles for a fair comparison. We do not compare DynNPC with other approaches due to differences in experimental configurations that can not be fully reproduced, or because they have been compared by our selected baselines. DoppelTest [32] aims to find EIVs by bridging multiple ADSs together to find violations, which does not use NPCs in the simulator. We do not compare DynNPC with DoppelTest because it does not work natively with LGSVL and needs to run multiple instances of Apollo concurrently, increasing the computational cost. All comparisons with baselines show statistically significant differences ($p = 0.0079$) on the metrics by Mann-Whitney U test, and the Cohen's d is far greater than 0.8.

Last, the subjective diagnosis and classification of violations induced by the Ego vehicle affects the validity. To mitigate this threat, we use DiaVio [48], an LLM-empowered diagnosis approach, and ACAV [64], a causality-based analysis tool. After obtaining the preliminary diagnosis results from DiaVio, we asked two additional authors to independently inspect each confirmed EIV and classify it according to the predefined criteria in Sec. 4.1, namely violation type, NPC maneuver category, and underlying causal factor. For collision violations, ACAV was additionally used to support root-cause analysis. When disagreements arose, a third author joined the discussion to reach a consensus. The resulting Cohen's kappa coefficient is 0.862, indicating a high level of inter-rater agreement.

6 Related Work

Scenario-based testing [18, 28, 38, 44, 46, 74, 77] has been widely studied to generate diverse driving scenarios for ADS testing to identify safety violations. A scenario-based safety evaluation framework has been proposed in ISO 34502 [33]. Some domain specific languages [4, 5, 22, 59, 60] have been proposed to describe the driving scenarios. A few works [8, 17, 23, 26, 66, 73] attempt to reproduce real-world data (e.g., traffic accident reports and vehicle trajectories) to find corner cases in simulation. To accelerate the identifying of violations, numerous search-based works [1, 9, 43, 67, 68] use a genetic algorithm-based approach to generate scenarios where the Ego vehicle may collide with NPC vehicles, while a few works [2, 15, 25, 31, 37, 49] guide the ADSs to violate predefined rules, such as failing to reach their destinations, or to exhibit incorrect behaviors like speeding or executing unsafe lane changes. Some works [42, 45, 65, 75] propose to generate driving scenarios where ADSs break specific traffic rules. Additionally, Lu et al. [47], Zhong et al. [76] and Wang et al. [70] employ neural network to guide the generation of scenarios. Besides, several works [10, 55, 72] investigate the metrics (e.g., physical environment-state coverage metric [29]) in simulation to guide the search and Chen et al. [14] study the configuration of simulation in ADS testing.

However, many prior scenario-based testing approaches, especially search-based ones, determine most scenario elements before execution, including NPC behaviors. They may insufficiently capture interactions conditioned on the Ego vehicle's real-time behavior and traffic signals during simulation, resulting in NPC vehicles not obeying traffic rules (*e.g.,* traffic lights) and performing unreasonable behaviors. In contrast, DynNPC dynamically generates NPC maneuvers and trajectories during execution, which helps find more violations induced by the ADS. Huai et al. [32] also aim to maximize violations induced by the ADS. They bridge multiple ADSs for interaction using Apollo's own simulation module *SimControl*, rather than using NPC vehicles in the simulator. However, this is achieved at the cost of feeding ground truth directly into the localization and perception modules of Apollo. Differently, we propose a new framework to generate scenarios in the simulator and test ADSs at the system-level.

Several recent works further explore reinforcement learning to generate interactive and adversarial NPC behaviors during execution [19, 61, 63]. Unlike traditional search-based approaches that manipulate static scenario elements, these works model surrounding vehicles as independent adversarial agents and train them via reinforcement learning to actively challenge the Ego vehicle during simulation, maximizing the likelihood of exposing violation scenarios. For example, Wang et al. [71] propose an adversarial multi-agent framework that generates attack-oriented traffic behaviors to efficiently uncover safety violations. However, these approaches usually rely on learned policies and carefully designed rewards, which require substantial training data and computational resources. Differently, DynNPC adopts a rule-based mechanism that does not require policy training, providing controllable and interpretable NPC behaviors. Moreover, DynNPC tightly integrates runtime NPC behavior generation with search-based scenario generation and execution over driving tasks, weather conditions, and traffic signal configurations, enabling effective and efficient system-level testing for finding more violations induced by the ADS.

7 Conclusion

We have proposed and implemented DynNPC, a novel search-based testing framework, to find more violation scenarios induced by the ADS through dynamically generating NPC vehicle behaviors during simulation execution. Large-scale experiments have been conducted to demonstrate the effectiveness and efficiency of DynNPC. In future, we plan to extend DynNPC to support more ADSs and simulators. Moreover, we also plan to support more types of traffic participants (*i.e.,* pedestrians and cyclists). The experimental records and source code of our work is available at our replication site [24].

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant No. 92582205).

References

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

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

[3] National Highway Traffic Safety Administration. 2016. *NHTSA Crash Viewer.* Retrieved February 15, 2025 from https://www.nhtsa.gov/

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

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

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

[7] Baidu. 2022. *Apollo: An open autonomous driving platform.* Retrieved January 12, 2025 from https://github.com/ApolloAuto/apollo

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

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

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

[11] Julian Bock, Robert Krajewski, Tobias Moers, Steffen Runde, Lennart Vater, and Lutz Eckstein. 2020. The ind dataset: A drone dataset of naturalistic road user trajectories at german intersections. In *2020 IEEE Intelligent Vehicles Symposium*. 1929–1934.

[12] Assaf Botzer, Oren Musicant, and Yaniv Mama. 2019. Relationship between hazard-perception-test scores and proportion of hard-braking events during on-road driving–An investigation using a range of thresholds for hard-braking. *Accident Analysis & Prevention* 132 (2019), 105267.

[13] California Department of Motor Vehicles. 2026. California Driver Handbook: Laws and Rules of the Road. https://www.dmv.ca.gov/portal/handbook/california-driver-handbook/laws-and-rules-of-the-road/. Accessed: 2026-04-05.

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

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

[16] Jacob Cohen. 2013. *Statistical power analysis for the behavioral sciences.* routledge.

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

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

[19] Andréa Doreste, Matteo Biagiola, and Paolo Tonella. 2024. Adversarial testing with reinforcement learning: A case study on autonomous driving. In *Proceedings of the 2024 IEEE Conference on Software Testing, Verification and Validation*. 293–304.

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

[21] Federal Highway Administration. 2003. Manual on Uniform Traffic Control Devices, Chapter 4D: Traffic Control Signal Features. https://mutcd.fhwa.dot.gov/htm/2003r1/part4/part4d.htm. Accessed: 2026-04-05.

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

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

[24] Anonymous Github. 2026. *DynNPC.* Retrieved April 3, 2026 from https://anonymous.4open.science/r/DynNPC/README.md

[25] Christoph Gladisch, Thomas Heinz, Christian Heinzemann, Jens Oehlerking, Anne von Vietinghoff, and Tim Pfitzer. 2020. Experience Paper: Search-Based Testing in Automated Driving Control Applications. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*. 26–37.
[26] An Guo, Yuan Zhou, Haoxiang Tian, Chunrong Fang, Yunjian Sun, Weisong Sun, Xinyu Gao, Anh Tuan Luu, Yang Liu, and Zhenyu Chen. 2024. Sovar: Build generalizable scenarios from accident reports for autonomous driving testing. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 268–280.
[27] Richard W Hamming. 1986. *Coding and information theory*. Prentice-Hall, Inc.
[28] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel C. Briand. 2020. Comparing Offline and Online Testing of Deep Neural Networks: An Autonomous Car Case Study. In *Proceedings of the IEEE 13th International Conference on Software Testing, Validation and Verification*. 85–95.
[29] Carl Hildebrandt, Meriel von Stein, and Sebastian Elbaum. 2023. PhysCov: physical test coverage for autonomous vehicles. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 449–461.
[30] Yuqi Huai. 2023. *SORA-SVL*. Retrieved January 12, 2025 from https://www.ics.uci.edu/~yhuai/SORA-SVL/
[31] Y. Huai, S. Almanee, Y. Chen, X. Wu, Q. Chen, and J. Garcia. 2023. scenoRITA: Generating Diverse, Fully Mutable, Test Scenarios for Autonomous Vehicle Planning. *IEEE Transactions on Software Engineering* 49, 10 (2023), 4656–4676.
[32] Yuqi Huai, Yuntianyi Chen, Sumaya Almanee, Tuan Ngo, Xiang Liao, Ziwen Wan, Qi Alfred Chen, and Joshua Garcia. 2023. Doppelgänger Test Generation for Revealing Bugs in Autonomous Driving Software. In *Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering*. 2591–2603.
[33] ISO. 2022. *ISO 34502:2022 - Road vehicles — Test scenarios for automated driving systems — Scenario based safety evaluation framework*. Retrieved July 3, 2025 from https://www.iso.org/standard/78951.html
[34] Nidhi Kalra and Susan M Paddock. 2016. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice* 94 (2016), 182–193.
[35] Srishti Khemka. 2021. Incident 347: Waymo Self-Driving Taxi Behaved Unexpectedly, Driving away from Support Crew. In *AI Incident Database*, Khoa Lam (Ed.). Responsible AI Collaborative. Retrieved February 13, 2023 from https://incidentdatabase.ai/cite/347/
[36] Rebecca Killick, Paul Fearnhead, and Idris A Eckley. 2012. Optimal detection of changepoints with a linear computational cost. *J. Amer. Statist. Assoc.* 107, 500 (2012), 1590–1598.
[37] Seulbae Kim, Major Liu, Junghwan" John" Rhee, Yuseok Jeon, Yonghwi Kwon, and Chung Hwan Kim. 2022. Drivefuzz: Discovering autonomous driving bugs through driving quality-guided fuzzing. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 1753–1767.
[38] Moritz Klischat and Matthias Althoff. 2022. Falsifying motion plans of autonomous vehicles with abstractly specified traffic scenarios. *IEEE Transactions on Intelligent Vehicles* 8, 2 (2022), 1717–1730.
[39] Lesson-8. 2006. *Federal Highway Administration University Course on Bicycle and Pedestrian Transportation*. Retrieved February 12, 2025 from https://www.fhwa.dot.gov/publications/research/safety/pedbike/05085/chapt8.cfm
[40] LGSVL. 2021. *lgsvl - A Python API for SVL Simulator*. Retrieved January 12, 2025 from https://github.com/lgsvl/PythonAPI
[41] LGSVl. 2021. *SVL Simulator: An Autonomous Vehicle Simulator*. Retrieved January 12, 2025 from https://github.com/lgsvl/simulator/releases/tag/2021.3
[42] Changwen Li, Joseph Sifakis, Qiang Wang, Rongjie Yan, and Jian Zhang. 2023. Simulation-Based Validation for Autonomous Driving Systems. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 842–853.
[43] Guanpeng Li, Yiran Li, Saurabh Jha, Timothy Tsai, Michael Sullivan, Siva Kumar Sastry Hari, Zbigniew Kalbarczyk, and Ravishankar Iyer. 2020. Av-fuzzer: Finding safety violations in autonomous driving systems. In *Proceedings of the 2020 IEEE 31st International Symposium on Software Reliability Engineering*. 25–36.
[44] Li Li, Wu-Ling Huang, Yuehu Liu, Nan-Ning Zheng, and Fei-Yue Wang. 2016. Intelligence testing for autonomous vehicles: A new approach. *IEEE Transactions on Intelligent Vehicles* 1, 2 (2016), 158–166.
[45] Zhongrui Li, Jiarun Dai, Zongan Huang, Nianhao You, Yuan Zhang, and Min Yang. 2024. VioHawk: Detecting Traffic Violations of Autonomous Driving Systems through Criticality-Guided Simulation Testing. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 844–855.
[46] Guannan Lou, Yao Deng, Xi Zheng, Mengshi Zhang, and Tianyi Zhang. 2022. Testing of autonomous driving systems: where are we and where should we go?. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 31–43.
[47] Chengjie Lu. 2023. Test Scenario Generation for Autonomous Driving Systems with Reinforcement Learning. In *Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings*. 317–319.
[48] You Lu, Yifan Tian, Yuyang Bi, Bihuan Chen, and Xin Peng. 2024. Diavio: Llm-empowered diagnosis of safety violations in ads simulation testing. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 376–388.
[49] Yixing Luo, Xiao-Yi Zhang, Paolo Arcaini, Zhi Jin, Haiyan Zhao, Fuyuki Ishikawa, Rongxin Wu, and Tao Xie. 2021. Targeting requirements violations of autonomous driving systems by dynamic evolutionary search. In *Proceedings of the 2021 36th IEEE/ACM International*

Conference on Automated Software Engineering. 279–291.
[50] Interactive Mathematics. 2024. *Velocity (s-t) Graphs.* Retrieved August 12, 2024 from https://www.intmath.com/kinematics/1-velocity-graphs.php
[51] Sean McGregor. 2018. Incident 4: Uber AV Killed Pedestrian in Arizona. In *AI Incident Database*, Sean McGregor (Ed.). Responsible AI Collaborative. Retrieved February 13, 2023 from https://incidentdatabase.ai/cite/4
[52] Patrick E McKnight and Julius Najab. 2010. Mann-Whitney U Test. *The Corsini encyclopedia of psychology* (2010), 1–1.
[53] Michael E Mortenson. 1999. *Mathematics for computer graphics applications.* Industrial Press Inc.
[54] Wassim G Najm, Raja Ranganathan, Gowrishankar Srinivasan, John D Smith, Samuel Toma, Elizabeth D Swanson, August Burgett, et al. 2013. *Description of light-vehicle pre-crash scenarios for safety applications based on vehicle-to-vehicle communications.* Technical Report. United States. Department of Transportation. National Highway Traffic Safety.
[55] Neelofar Neelofar and Aldeida Aleti. 2024. Towards reliable ai: Adequacy metrics for ensuring the quality of system-level testing of autonomous vehicles. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–12.
[56] United States Department of Transportation Federal Highway Administration. 2021. *Traffic Signal Timing Manual.* Retrieved February 17, 2025 from https://ops.fhwa.dot.gov/publications/fhwahop08024/chapter4.htm
[57] National Committee on Uniform Traffic Laws. 1952. *Uniform vehicle code*. Vol. 5. Department of Commerce, Bureau of Public Roads.
[58] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. 2016. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles* 1, 1 (2016), 33–55.
[59] Rodrigo Queiroz, Thorsten Berger, and Krzysztof Czarnecki. 2019. GeoScenario: An Open DSL for Autonomous Driving Scenario Representation. In *Proceedings of the IEEE Intelligent Vehicles Symposium*. 287–294.
[60] Rodrigo Queiroz, Divit Sharma, Ricardo Caldas, Krzysztof Czarnecki, Sergio García, Thorsten Berger, and Patrizio Pelliccione. 2024. A driver-vehicle model for ADS scenario-based testing. *IEEE Transactions on Intelligent Transportation Systems* 25, 8 (2024), 8641–8654.
[61] Davis Rempe, Jonah Philion, Leonidas J Guibas, Sanja Fidler, and Or Litany. 2022. Generating useful accident-prone driving scenarios via a learned traffic prior. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 17305–17315.
[62] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, et al. 2020. LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving. *arXiv preprint arXiv:2005.03778* (2020).
[63] Luke Rowe, Roger Girgis, Anthony Gosselin, Bruno Carrez, Florian Golemo, Felix Heide, Liam Paull, and Christopher Pal. 2024. CtRL-Sim: Reactive and Controllable Driving Agents with Offline Reinforcement Learning. In *Proceedings of The 8th Conference on Robot Learning, in Proceedings of Machine Learning Research*. 3600–3621.
[64] Huijia Sun, Christopher M Poskitt, Yang Sun, Jun Sun, and Yuqi Chen. 2024. ACAV: a framework for automatic causality analysis in autonomous vehicle accident recordings. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.
[65] Yang Sun, Christopher M Poskitt, Jun Sun, Yuqi Chen, and Zijiang Yang. 2022. LawBreaker: An approach for specifying traffic laws and fuzzing autonomous vehicles. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.
[66] Shuncheng Tang, Zhenya Zhang, Jixiang Zhou, Lei Lei, Yuan Zhou, and Yinxing Xue. 2024. LeGEND: A Top-Down Approach to Scenario Generation of Autonomous Driving Systems Assisted by Large Language Models. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 1497–1508.
[67] Haoxiang Tian, Yan Jiang, Guoquan Wu, Jiren Yan, Jun Wei, Wei Chen, Shuo Li, and Dan Ye. 2022. MOSAT: finding safety violations of autonomous driving systems using multi-objective genetic algorithm. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 94–106.
[68] Haoxiang Tian, Guoquan Wu, Jiren Yan, Yan Jiang, Jun Wei, Wei Chen, Shuo Li, and Dan Ye. 2022. Generating Critical Test Scenarios for Autonomous Driving Systems via Influential Behavior Patterns. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.
[69] Simon Ulbrich, Till Menzel, Andreas Reschka, Fabian Schuldt, and Markus Maurer. 2015. Defining and substantiating the terms scene, situation, and scenario for automated driving. In *2015 IEEE 18th international conference on intelligent transportation systems*. IEEE, 982–988.
[70] Tong Wang, Taotao Gu, Huan Deng, Hu Li, Xiaohui Kuang, and Gang Zhao. 2024. Dance of the ADS: Orchestrating Failures through Historically-Informed Scenario Fuzzing. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 1086–1098.
[71] Tiexin Wang, Shuo Tian, Gulent Asalif Minas, and Chunyang Bian. 2025. AMACollision: An advanced framework for testing autonomous vehicles based on adversarial multi-agent. *Journal of Systems and Software* (2025), 112578.
[72] Trey Woodlief, Felipe Toledo, Sebastian Elbaum, and Matthew B Dwyer. 2024. S3C: Spatial Semantic Scene Coverage for Autonomous Vehicles. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.
[73] Xudong Zhang and Yan Cai. 2023. Building critical testing scenarios for autonomous driving from real accidents. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 462–474.
[74] Xinhai Zhang, Jianbo Tao, Kaige Tan, Martin Törngren, José Manuel Gaspar Sánchez, Muhammad Rusyadi Ramli, Xin Tao, Magnus Gyllenhammar, Franz Wotawa, Naveen Mohan, Mihai Nica, and Hermann Felbinger. 2023. Finding Critical Scenarios for Automated

Driving Systems: A Systematic Mapping Study. *IEEE Transactions on Software Engineering* 49, 3 (2023), 991–1026.

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

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

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