Decictor: Towards Evaluating the Robustness of Decision-Making in Autonomous Driving Systems

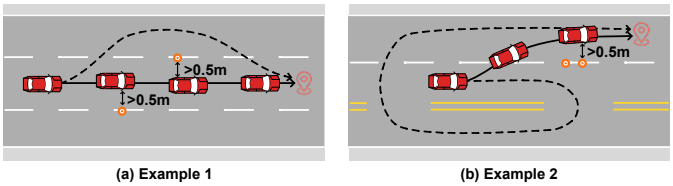
Mingfei Cheng[1], Xiaofei Xie[1], Yuan Zhou[2*], Junjie Wang[3], Guozhu Meng[4] and Kairui Yang[5]
[1]Singapore Management University, Singapore [2]Zhejiang Sci-Tech University, China
[3]Tianjin University, China [4]Chinese Academy of Sciences, China [5]Alibaba Group, China

***Abstract*—Autonomous Driving System (ADS) testing is crucial in ADS development, with the current primary focus being on safety. However, the evaluation of non-safety-critical performance, particularly the ADS's ability to make optimal decisions and produce optimal paths for autonomous vehicles (AVs), is also vital to ensure the intelligence and reduce risks of AVs. Currently, there is little work dedicated to assessing the robustness of ADSs' path-planning decisions (PPDs), i.e., whether an ADS can maintain the optimal PPD after an insignificant change in the environment. The key challenges include the lack of clear oracles for assessing PPD optimality and the difficulty in searching for scenarios that lead to non-optimal PPDs. To fill this gap, in this paper, we focus on evaluating the robustness of ADSs' PPDs and propose the first method, *Decictor*, for generating non-optimal decision scenarios (NoDSs), where the ADS does not plan optimal paths for AVs. *Decictor* comprises three main components: Non-invasive Mutation, Consistency Check, and Feedback. To overcome the oracle challenge, Non-invasive Mutation is devised to implement conservative modifications, ensuring the preservation of the original optimal path in the mutated scenarios. Subsequently, the Consistency Check is applied to determine the presence of non-optimal PPDs by comparing the driving paths in the original and mutated scenarios. To deal with the challenge of large environment space, we design Feedback metrics that integrate spatial and temporal dimensions of the AV's movement. These metrics are crucial for effectively steering the generation of NoDSs. Therefore, *Decictor* can generate NoDSs by generating new scenarios and then identifying NoDSs in the new scenarios. We evaluate *Decictor* on Baidu Apollo, an open-source and production-grade ADS. The experimental results validate the effectiveness of *Decictor* in detecting non-optimal PPDs of ADSs. It generates 63.9 NoDSs in total, while the best-performing baseline only detects 35.4 NoDSs.**

I. Introduction

Autonomous Driving Systems (ADSs) have been a revolutionary technology with the potential to transform our transportation system into an intelligent one. ADSs aim to enable vehicles to operate without human intervention, relying on a combination of different sensors (e.g., camera, radar, lidar, and GPS) and artificial intelligence algorithms to perceive the environment, make decisions, and navigate safely. Even though the development of ADSs has seen significant progress over the past few decades, it is still a great challenge to guarantee that the ADSs can satisfy all performance requirements under different situations due to the existing vulnerabilities in ADSs [1]. Therefore, before their real-world deployment, ADSs should be sufficiently tested in all aspects [2].

Usually, the requirements of ADSs can be classified as safety-critical and non-safety-critical. Safety-critical requirements are those essential for ensuring safe operations and performance of the autonomous vehicle. For example, an ADS should guarantee that the ego vehicle (i.e., the autonomous vehicle controlled by the ADS) should arrive at its destination without causing collisions or violating traffic rules. Non-safety-critical requirements refer to aspects that do not directly impact safety but are important for user experiences and optimal vehicle performance, such as motion efficiency, passenger comfort, and energy consumption.



(a) Example 1 (b) Example 2

Fig. 1. Illustration of non-optimal PPDs. **Solid lines** are expected optimal paths. **Dotted lines** are non-optimal paths planned by the ADS.

Currently, various testing technologies have been proposed to evaluate the safety-critical requirements of ADSs (referred to as "safety testing"), such as data-driven methods [3]–[6] and search-based methods [7]–[17]. They aim to generate safety-critical scenarios, under which the ego vehicle will cause safety issues, such as collisions and traffic rule violations. *To the best of our knowledge, there is limited research on evaluating the robustness of the path-planning decisions (PPDs) in ADSs, i.e., their ability to plan optimal paths in dynamic environments*. PPD optimality is an important non-safety-critical property for guaranteeing the motion efficiency of the ego vehicle in complex traffic scenarios, but it may not be achieved by ADSs. Fig. 1 presents two illustrative examples that highlight PPD robustness issues, where the ADS fails to determine the optimal paths after minor environmental changes, such as placing two small obstacles on the lane boundaries that would not affect the optimal path (i.e., solid lines). Note that, in these changes, the ego vehicle still maintains a safe distance [18] (i.e., more than 0.5 meters from the obstacle) to pass the scenarios if it follows the optimal paths. The video [19] shows an example that Tesla FSD cannot make robust PPDs at a fork in a real-world road.

An ADS with limited intelligence might inaccurately assess traffic scenarios, resulting in suboptimal choices. Such decisions could cause prolonged delays or select inefficient routes, leading to unnecessary hold-ups or even elevating safety risks. Therefore, evaluating the robustness of PPD of ADSs is critically vital, as it affects the comfort and efficiency of autonomous vehicles and could pose additional safety concerns. Note that, in this paper, PPD robustness evaluation aims to

* Corresponding author

1558-1225/25/$31.00 ©2025 IEEE
DOI 10.1109/ICSE55347.2025.00114

explore scenarios where safety issues are not present, instead focusing on assessing the optimal PPD capabilities of ADS within dynamic driving environments.

Technically, drawing on the foundational principles established by existing research in robustness evaluation research [20], [21], the core objective of robustness evaluation is to assess a model's capacity to maintain correct prediction following some perturbations. For instance, in image classification, an image is slightly modified (e.g., through the addition of noise) to test whether the model can still accurately classify the perturbed image. Gradient-based methods [20], [21] are commonly employed to efficiently calculate these perturbations. However, evaluating the PPD robustness of ADS presents unique challenges: ❶ Contrary to safety testing, which relies on explicit, measurable criteria (like collision detection or timeout occurrences) to identify safety-critical violations, the assessment of PPD optimality in ADS under new scenarios lacks clear oracles. While traditional robustness evaluations might control the extent of perturbations (e.g., the level of noise) to ensure the perturbed inputs match the truth labels of original inputs, the complexity inherent in AV scenarios complicates the control of scenario perturbations. ❷ The unpredictable and infinitely variable nature of driving conditions, influenced by diverse road users and environmental factors, complicates the search of "**N**on-**o**ptimal **D**ecision **S**cenarios" (NoDSs). The conventional gradient-based method is difficult to be applied in these scenarios due to discontinuities in the input space and the logical constraints that perturbations must obey, such as adherence to physical laws and ensuring that obstacles remain within realistic bounds. ❸ The evaluation of robustness typically requires a method to check prediction consistency between the original input and its perturbed counterpart. In classification tasks, the consistency check is straightforwardly verified by comparing their predicted labels. However, in ADS, decisions are represented by planned paths, which introduces complexity into the process of determining their consistency.

To address these challenges, we propose a novel testing method, called *Decictor*, comprising three main components *Non-invasive Mutation*, *Feedback* and *Consistency Check*. The innovative approach enables the effective and efficient identification of non-optimal PPDs and their associated NoDSs. Specifically, to address the first challenge ❶, we design a *non-invasive mutation* that conservatively modifies the behaviors of other participants within the driving environment in a way that the modifications would not affect the originally optimal path in the given "**O**ptimal **D**ecision **S**cenario" (ODS). For the second challenge ❷, we propose a fitness function designed to guide the generation of NoDSs. The function scores are based on two key metrics: firstly, it seeks to maximize the discrepancy in the driving paths between the ego vehicle in both the mutated and original scenarios, directly accentuating the non-optimal behaviors. Secondly, it assesses the variances between the scenarios from a broader perspective of ego behavior, measured by a vector encompassing velocity, acceleration, and orientation. The second metric is crucial as the behavior of the ego vehicle may directly affect its driving path. For the third challenge ❸, given the possible slight variations between the new driving path (in the mutated scenario) and the initially optimal path (in ODS), the direct path-level comparison is not reliable. To resolve this, we introduce an abstraction-based behavior comparison method. This technique evaluates the driving patterns in both scenarios through abstraction, where significant differences in routes indicate deviations from optimal PPDs by the ADS.

We have evaluated *Decictor* on the Baidu Apollo with its built-in SimControl [22]. We compared *Decictor* with nine baseline methods: two random methods, i.e., Random, which generates scenarios randomly, and Random-δ, which applies our non-invasive mutation and random selection to generate scenarios, along with seven state-of-the-art ADS testing methods, i.e., AVFuzzer [8], SAMOTA [10], BehAVExplor [9], DriveFuzz [13], scenoRITA [14], DoppelTest [15] and DeepCollision [17]. The evaluation results reveal the effectiveness and efficiency of *Decictor* in detecting NoDSs, demonstrating the path-planning robustness issues within the ADS. For example, *Decictor* generates a total of 63.9 NoDSs while the best-performing baseline only detects 35.4 NoDSs. Further experimental results demonstrate the usefulness of our mutation operation and feedback mechanisms meticulously designed within *Decictor*.

In summary, this paper makes the following contributions:

1) We are the first to investigate the problem of non-optimal PPDs made by ADSs. Our work provides valuable and original insights into evaluating the non-safety-critical performance of ADSs.
2) We develop a search-based method that efficiently generates NoDSs and evaluates the PPD robustness of ADSs.
3) We conduct extensive experiments to evaluate the effectiveness and usefulness of *Decictor* on Baidu Apollo. As a result, a total of 63.9 NoDSs is discovered on six initial ODSs on average, demonstrating the potential and value of *Decictor* in detecting non-optimal PPDs made by ADSs.

II. Background and Notation

A. Autonomous Driving Systems

ADSs control the motion of autonomous vehicles. Existing ADSs mainly contain two categories: End-to-End (E2E) systems [23]–[25], and module-based ADSs [22], [26], [27]. E2E systems use united deep learning models to generate control decisions from sensor data directly. Recently, the rapid development of Deep Learning have led to high-performance E2E systems in close-loop datasets, such as OpenPilot [25]. However, these E2E systems still perform poorly on unseen data. In contrast, module-based ADSs have better performance in various scenarios. A typical module-based ADS, such as Baidu Apollo [22], usually consists of localization, perception, prediction, planning, and control modules to generate decisions from rich sensor data. The localization module provides the location of the ego vehicle by fusing multiple input data from GPS, IMU, and LiDAR sensors. The perception module takes camera images, LiDAR point clouds, and Radar signals as

inputs to detect the surrounding environment (e.g., traffic lights) and objects (e.g. other vehicles) by mainly using deep neural networks. The prediction module is responsible for tracking and predicting the trajectories of all surrounding objects detected by the perception module. Given the results of perception and prediction modules, the planning module then generates a local collision-free trajectory for the ego vehicle. Finally, the control module converts the planned trajectory to vehicle control commands (e.g., steering, throttle, and braking) and sends them to the chassis of the vehicle. In this paper, we choose to test module-based ADSs on a simulation platform, where the ADS connects to a simulator via a communication bridge. Same to previous works [9], [14], [15], the ADS receives perfect perception data from the simulator, and sends the control commands to the ego vehicle in the simulator.

B. Scenario

ADS testing necessitates a collection of scenarios as inputs. Each scenario is characterized by a specific environment (e.g., road), the scenery and objects (e.g., static obstacles and Non-Player Character (NPC) vehicles). Complex scenarios are generated by combining relevant attributes from the Operational Design Domains (ODDs) [28]. Due to the vast attribute space, covering all attributes with all possible values is impractical. Existing studies [8]–[10], [13]–[17] address this by selecting specific subsets of attributes. In this paper, we evaluate the non-safety-critical performance of ADS motion using scenarios formulated with traffic cones as static obstacles and NPC vehicles as dynamic objects.

Therefore, a *scenario* can be described as a tuple $s = \{\mathcal{A}, \mathcal{P}\}$, where $\mathcal{A}$ is the motion task of the ADS under test, including the start position and the destination, $\mathcal{P}$ is a finite set of participants, including the set of static obstacles and dynamic NPC vehicles. Note that in our paper, participants in the scenario do not include the ego vehicle controlled by the ADS. Scenario observation is a sequence of scenes within the execution of the scenario, and each scene represents the states of the ego vehicle and other participants at a timestamp. Formally, given a scenario $s = \{\mathcal{A}, \mathcal{P}\}$, its observation is denoted as $O(s) = \{o_0, o_1, \ldots, o_k\}$, where k is the length of the observation and o_i is a scene at timestamp i. In detail, $o_i = \{y_i^0, y_i^1, \ldots, y_i^{|\mathcal{P}|}\}$ where $y_i^j = \{p_i^j, \theta_i^j, v_i^j, a_i^j\}$ denotes the waypoint of a participant $j \in \mathcal{P}$ at timestamp i, including the center position p_i^j, the heading θ_i^j, the velocity v_i^j and the acceleration a_i^j. A driving path of the participant j can be defined as $\tau^j(s) = \{p_0^j, \ldots, p_k^j\}$. By default, we use $\tau(s)$ to represent the driving path of the ego vehicle in the scenario s. Unless otherwise specified, the driving path in the following context refers to the path of the ego vehicle in s, i.e., $\tau(s)$.

III. Overview

A. Problem Definition

This paper aims to assess the optimization capability of the path-planning process in ADSs. Given an ODS s and its associated optimal path for the ego vehicle $\tau(s)$, PPD optimality testing aims to assess the robustness of ADS's PPD, i.e., its ability to consistently navigate along $\tau(s)$. Following the robustness definitions of neural networks [29]–[32], we define the PPD robustness as follows:

Definition 1 (Path-planning Decision Robustness)**.** *Given an ODS $s = \{\mathcal{A}, \mathcal{P}\}$, we formalize the path-planning decision (PPD) robustness as:*

$$\forall s'.||s - s'||_\delta = True \implies \epsilon(\tau(s), \tau(s')) = True \quad (1)$$

where $||\cdot||_\delta$ is used to constrain scenario mutation, ensuring that the mutation δ modifies a subset of participants $P \subseteq \mathcal{P}$ in s to produce s', without compromising the optimality of the original driving path, i.e., $||s - s'||_\delta = True$. The function ϵ serves as a consistency checking mechanism, evaluating whether the two driving paths executed by the ADS in s and s' are equivalent.

The definition of PPD robustness stipulates that, following minor perturbations (i.e., $s' = \delta(s, P)$), the ADS should maintain the capacity to select the original optimal driving path (i.e., $\pi(s)$). *The main goal of the PPD robustness testing is to identify scenarios where the ADS can accomplish its task without encountering safety issues, yet does not manage to plan and execute the optimal driving path.* Note that the PPD optimality of the seed ODS s can be manually confirmed while the PPD optimality of mutated scenarios s' can be automatically checked by evaluating their consistency.

B. Approach Overview

Fig. 2 provides a high-level depiction of our testing framework *Decictor*, designed to detect non-optimal PPDs. The basic idea underlying our method involves generating a scenario s', where the ego vehicle opts for a path far from the optimal one in the ODS s under the non-invasive mutation δ. It can be formulated as an optimization problem aiming to maximize the difference between the driving paths of s and s':

$$\Delta_s = \arg\max_{\Delta} \mathcal{D}(\tau(s), \tau(s')), \text{ where } s' = \delta(s, \Delta) \quad (2)$$

where Δ symbolizes the non-invasive perturbations on the scenario s, and $\mathcal{D}$ is a function to measure the distance or difference between the driving paths of the two scenarios.

Decictor adopts a search-based method to solve the optimization problem. The process initiates with a seed ODS s^* and aims to output a set of violation tests, i.e., NoDSs, considering the optimal path in s^*. We first initialize a population $\mathbf{Q}$ based on s^*. *Decictor* then optimizes this population iteratively until the testing budget B exceeds. In each iteration, *Decictor* performs *Non-invasive Mutation* (δ) to generate the offspring $\mathbf{Q}'$ that has the same size as $\mathbf{Q}$. The mutation strategy ensures that the original optimal path in s^* remains optimal in the new scenarios. To determine if the ADS makes the optimal PPDs in these new scenarios, we introduce *Consistency Check* (ϵ), a hierarchical approach for measuring the equivalence with regard to PPD optimality between the seed ODS and these new scenarios. *Consistency Check* initially monitors whether a new scenario completes the task during simulation execution. If the

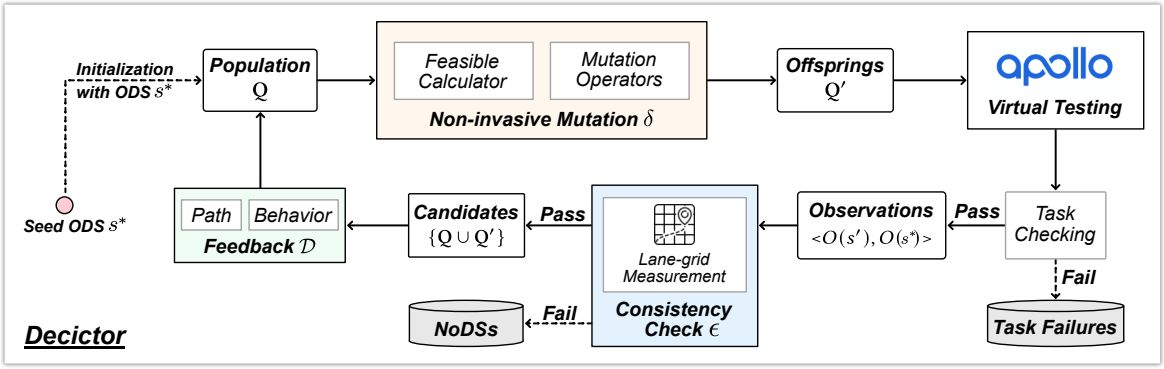


Fig. 2. Overview of *Decictor*.

Algorithm 1: *Decictor* Algorithm

Input : The seed ODS: s^*
Output : The set of NoDSs: $\mathbf{F}_n$
Parameters: Population size N, Testing budget B

1 $\mathbf{F}_n \leftarrow \{\}$, $\mathbf{Q} \leftarrow \{\}$
2 **for** $i \in \{1, \ldots, N\}$ **do**
3 $\mathbf{Q} \leftarrow \mathbf{Q} \cup \{s^*\}$
4 **repeat**
5 $\mathbf{Q}' \leftarrow \{\}$
6 **for** $\hat{s} \in \mathbf{Q}$ **do**
7 $s' \leftarrow \delta(\hat{s}, \Delta)$ `// Non-invasive Mutation`
8 $r_{tc}, O(s') \leftarrow$ **SimExecution**(s')
9 **if** r_{tc} *is passed* **then**
 `// Consistency Check`
10 $r_{cc} \leftarrow \epsilon(\tau(s^*), \tau(s'))$
11 **if** r_{cc} *is passed* **then**
12 $\mathbf{Q}' \leftarrow \mathbf{Q}' \cup \{s'\}$
13 **else**
14 $\mathbf{F}_n \leftarrow \mathbf{F}_n \cup \{s'\}$
15 $\mathbf{Q} \leftarrow$ **Selection**$(\{\mathbf{Q} \cup \mathbf{Q}'\}, N)$ `// Feedback`
16 **until** *Testing budget* B *exhausted*;
17 **return** $\mathbf{F}_n$

modified scenario results in a task failure, such as collisions, it is not considered as a NoDS of interest. This is because such failures can already be detected by existing AV testing tools that utilize timeout or collision detection oracles. Conversely, if the new scenario passes the task checking, we collect the observation from the simulator. The *Consistency Check* then verifies the equivalence of the driving paths between the seed ODS and the new scenario using a lane-grid measurement. Detection of the consistency violation in this step identifies a NoDS. If no violation occurs, *Decictor* proceeds to select the superior population from the candidates of the old and new populations for the next iteration. This selection process is guided by the Feedback ($\mathcal{D}$) that takes into account the spatial and temporal characteristics, i.e., the driving path and the behaviors, of the ego vehicle's motion.

IV. Approach

Algorithm 1 presents the main algorithmic procedure of *Decictor*. *Decictor* receives a seed ODS s^* as the input and outputs a set of NoDSs. Parameters N and B can be adjusted to configure the population size and testing budget, respectively. The algorithm begins by creating an initial population with the given seed ODS s^* (Lines 2-3). In each iteration, the algorithm first generates the offspring by mutating each scenario in the population $\mathbf{Q}$ (Lines 6-7). Each new scenario s' is first executed in the simulation platform consisting of a simulator and the ADS under test (Line 8) and verified by the task checking r_{tc} (Line 9). The new scenario passed the task checking is then evaluated for PPD consistency (Lines 10-14). If violated, a NoDS is identified and added to the set $\mathbf{F}_n$ (Line 14). If no violation is detected, the new scenario is added to the offspring $\mathbf{Q}'$. The scenario selection picks the top N scenarios from the union of $\mathbf{Q}$ and $\mathbf{Q}'$ based on their fitness scores (Line 15). The algorithm ends by returning detected NoDSs $\mathbf{F}_n$ (Line 17).

In the following sections, we introduce the key components of *Decictor*: the *Non-invasive Mutation* (δ), the *Consistency Check* (ϵ), and the *Feedback* ($\mathcal{D}$).

A. Non-invasive Mutation δ

The main challenge of the mutation is to ensure that in the mutated scenario, the original optimal path of the ego vehicle in the seed ODS is not affected. Following existing mutation techniques for ADS safety testing [8]–[10], [33], we only alter the waypoints of NPC vehicles or other participants, leaving the ego vehicle unchanged. However, the primary challenge is that these mutations could significantly impact the ego vehicle's original optimal path. To mitigate this challenge, we introduce a conservative mutation approach, namely the non-invasive mutation strategy, designed to create new scenarios with less impact on the original optimal path. While it may be theoretically difficult, or even impossible, to ensure that changes do not affect the ego vehicle's original optimal path, our method adopts a conservative approach. This involves calculating "safe zones" for mutation, where adjustments to participants' behaviors are unlikely to influence the original optimal path. This concept is akin to existing adversarial attack methodologies, where the difference between the new and original inputs is limited within an L_P norm boundary [32]. Such constraints are intended to preserve the semantic integrity of the input, but a theoretical guarantee of no impact is challenging to establish [21], [34].

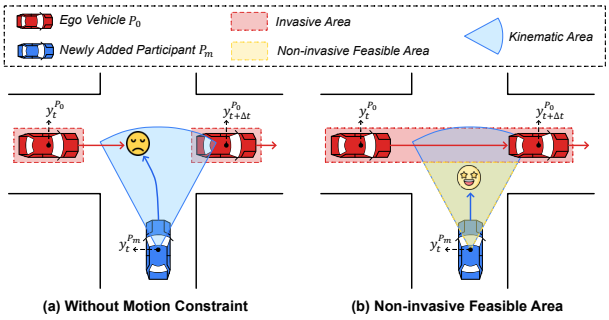


Fig. 3. Illustrative example of feasible area calculation.

The non-invasive mutation mainly includes *adding* new participants within the non-invasive feasible areas, *removing* existing added participants and *changing* existing participants.

Non-invasive Feasible Area. To minimize the impact of the mutations on the original optimal path $\pi(s^*)$, for any ODS $\hat{s} = \{\mathcal{A}, \mathcal{P}\}$ derived from the seed ODS s^*, we initially calculate a set of non-invasive feasible areas. These areas are designated so that the introduction of a new participant P_m to $\hat{s}$ is unlikely to influence the optimal path $\pi(s^*)$ and other participants $\mathcal{P}$ in $\hat{s}$ if P_m navigates within these non-invasive zones. Note that reducing the influence on the motions of other participants is also important, as their behaviors could indirectly influence the driving path of the ego vehicle.

The determination of the feasible area at any given moment (t) must account for the effects of both time and space. Specifically, it involves ensuring that the ego vehicle's original optimal path $\pi(s^*)$ and the paths of $\mathcal{P}$ remain unaffected during a certain time frame (Δt) surrounding that moment. Suppose the current observation is $O(\hat{s}) = \{\hat{o}_0, \hat{o}_1, \ldots, \hat{o}_k\}$ with a sampling time step Δt, we define the non-invasive feasible area regarding to the object $p \in \{\mathcal{P} \cup P_0\}$ during $[t, t + \Delta t]$ as $R_p(\hat{s}, t, \Delta t)$, where P_0 is the ego vehicle.

Fig. 3 illustrates the basic idea about the computation of $R_p(\hat{s}, t, \Delta t)$, taking the ego vehicle as an example. We first estimate the feasible motion area of the new participant, denoted as $R(y_t)$. Specifically, given the motion constraints (i.e., speed and steering ranges) and the position y_t at timestamp t, we can calculate a sector range in Fig. 3(a) according to the kinematic model [35]. Second, we estimate the motion area of the ego vehicle P_0 between $[t, t + \Delta t]$ based on the original optimal path $\pi(s^*)$, which is denoted as $R^{P_0}(\pi(s^*), \Delta t)$, e.g., the dashed light red area in Fig. 3(b). Note that the motion area of the ego vehicle is directly calculated from the real movement recorded in the scenario observations, which can be regarded as a rectangle between t and $t + \Delta_t$. Intuitively, the new participant should not move into $R^{P_0}(\pi(s^*), \Delta t)$ during the time frame $[t, t + \Delta t]$ so as not to affect the optimal path of the ego vehicle.

Therefore, we can calculate the non-invasive feasible area (e.g., the light yellow area in Fig. 3(b)) as:

$$R_{P_0}(\hat{s}, t, \Delta t) = R(y_t) \setminus R^{P_0}(\pi(s^*), \Delta t), \tag{3}$$

Similarly, we can calculate the non-invasive feasible areas $R_p(\hat{s}, t, \Delta t)$ with respect to any participant $p \in \mathcal{P}$ based on its driving path $\pi^p(\hat{s})$. The inference of the non-invasive feasible area at timestamp $t + \Delta t$ should consider all newly added participants, which is calculated as:

$$R(\hat{s}, t, \Delta t) = \cap_{p \in \{\mathcal{P} \cup P_0\}} R_p(\hat{s}, t, \Delta t). \tag{4}$$

Note that the time step Δt will affect the computation of the non-invasive area. As Δt increases, the calculation of the non-invasive area becomes more conservative as the new participant does not affect the ego behavior in longer time.

Algorithm 2: Non-invasive Mutation Operators

Input : Participants in the seed ODS s^*: $\mathcal{P}_{s^*}$
The ego vehicle's initial optimal path: $\pi(s^*)$
Participants in the current ODS $\hat{s}$: $\mathcal{P}_{\hat{s}}$
Observation of $\hat{s}$: $O(\hat{s}) = \{\hat{o}_0, \hat{o}_1, \ldots, \hat{o}_k\}$
Output : Participants in the newly mutated scenario s': $\mathcal{P}_{s'}$
Parameters : Time step Δt between two successive waypoints.

1 **Function** `Adding()`:
2 $P_m \leftarrow \{\}$
3 $y_0 \leftarrow$ *a collision-free waypoint*
4 **for** $t \in \{0, 1, 2, \ldots, k-1\}$ **do**
5 $R(\hat{s}, t, \Delta t) \leftarrow$ `NonInvasiveArea`$(y_t, \pi(s^*), O(\hat{s}), \Delta t)$
6 **if** $R(\hat{s}, t, \Delta t)$ *is* $\emptyset$ **then**
7 **return** $\emptyset$
8 $y_{t+\Delta t} \leftarrow$ `Sample`$(R(\hat{s}, t, \Delta t))$
9 $P_m \leftarrow P_m \cup \{y_{t+\Delta t}\}$
10 $\mathcal{P}_{s'} \leftarrow \mathcal{P}_{\hat{s}} \cup \{P_m\}$
11 **return** $\mathcal{P}_{s'}$
12 **Function** `Removing()`:
13 $P_m \leftarrow$ `Sample`$(\mathcal{P}_{\hat{s}} \setminus \mathcal{P}_{s^*})$
14 $\mathcal{P}_{s'} \leftarrow \{\mathcal{P}_{\hat{s}} \setminus P_m\}$
15 **return** $\mathcal{P}_{s'}$
16 **Function** `Changing()`:
17 $\mathcal{P}_{s'} \leftarrow$ `Removing()`
18 $O(\hat{s}) \leftarrow O(\hat{s}) \setminus \mathbf{y}^{\{\mathcal{P}_{\hat{s}} \setminus \mathcal{P}_{s'}\}}$
19 $\mathcal{P}_{\hat{s}} \leftarrow \mathcal{P}_{s'}$
20 $\mathcal{P}_{s'} \leftarrow$ `Adding()`
21 **return** $\mathcal{P}_{s'}$

Mutation Operations. Algorithm 2 outlines the specific mutation operations: *Adding* (Lines 1-11), *Removing* (Lines 12-15) and *Changing* (Line 16-21). The non-invasive mutation takes as input $\mathcal{P}_{s^*}$ (the participants in the seed ODS s^*), $\mathcal{P}_{\hat{s}}$ (the participants in the current ODS $\hat{s}$) and $O(\hat{s})$ (the observation of $\hat{s}$). The observation consists of a sequence of scenes $\{\hat{s}_0, \ldots, \hat{s}_k\}$ with an equal time step Δt. In each iteration, the non-invasive mutation randomly chooses one operation to change participants in the scenario $\hat{s}$, resulting in a new scenario s'.

Adding. Adding operation aims to introduce complexity to the scenario by adding a new participant to $\hat{s}$ that is less likely to influence the optimal path. In detail, this operation initializes an empty waypoint set P_m and an initial collision-free waypoint y_0 (Lines 2-3). Waypoints for the added participant are iteratively generated (Lines 4-9), where each iteration involves

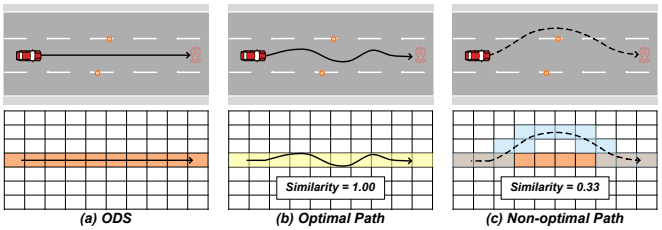


Fig. 4. Consistency Checking. **Above:** Driving paths. **Below:** Locations of the corresponding paths in the grid map.

calculating the non-invasive area (Line 5), sampling a non-invasive waypoint (Line 6-8), and adding it to P_m (Line 9). The process concludes by merging the resultant set with the original participants $\mathcal{P}_{\hat{s}}$ to form $\mathcal{P}_{s'}$ (Line 10-11).

Removing. Continuously adding new participants to the scenario will introduce too many obstacles, resulting in motion task failures. Thus, we introduce the removing operator. Note that only the participants in $\mathcal{P}_{\hat{s}} \setminus \mathcal{P}_{s^*}$, i.e., the participants newly added to the seed ODS s^*, can be removed, while the participants in the seed ODS will remain unchanged in all mutated scenarios. Because the removal of the participants in the seed ODS may affect the optimal path of the scenarios. Specifically, this operation removes a participant in $\mathcal{P}_{\hat{s}} \setminus \mathcal{P}_{s^*}$ randomly (Lines 13-14) and returns the mutated set (Line 15).

Changing. The changing operation aims to modify the scenario $\hat{s}$ by replacing an existing participant in $\mathcal{P}_{\hat{s}}$. This operation involves first removing an existing object from $\mathcal{P}\hat{s}$ (Line 17) and then adding a new one (Line 20). Specifically, we remove the observation of the removed participant $\mathbf{y}^{\{\mathcal{P}_{\hat{s}}}$ from $O(\hat{s})$ (Lines 18-19) to ensure that the calculation of the non-invasive feasible area does not include this participant.

B. Consistency Check ϵ

Given the seed ODS s^* and a mutated scenario s', we require a criterion ϵ to determine whether s' remains the optimal path in s^*. A direct approach might be checking if the driving path in s' (i.e., $\tau(s')$) and the original path of s^* (i.e., $\tau(s^*)$) are the same. However, even in real-world scenarios, human drivers do not strictly adhere to a straight line but exhibit some degree of zig-zag motion, as shown in Fig. 4(b). This motion can still be considered optimal even if it slightly deviates from the original path $\tau(s^*)$. Thus, we propose an abstraction-based method to quantify the similarity of the two driving paths.

Fig. 4 illustrates the basic idea of our method. We define an optimal area around the optimal path $\tau(s^*)$ in terms of the grid map. Any driving path falling or mostly falling within this area is considered optimal. For example, the orange area in Fig. 4(a) defines the optimal area of the driving path in the seed ODS. The driving path in Fig. 4(b) does not strictly adhere to the optimal path but remains within the optimal area, achieving the highest similarity and thus meeting the optimality criterion. In contrast, most of the path in Fig. 4(c) (i.e., the blue area) falls outside the optimal area, resulting in a low similarity and a non-optimal path.

Specifically, the comparison between the driving paths $\tau(s^*)$ and $\tau(s')$ is implemented by a grid-based approach, as shown in Fig. 4. The map is divided into grids, and each driving path is mapped to a set of grids. Technically, a driving path of the ego vehicle in a scenario s is represented as a sequence of locations $\tau(s) = \{p_0^0, \ldots, p_k^0\}$, where p_i^0 is the position at frame i. To collect the covered grids, a function g is used to map each position p to the grid where it is located. The covered grids for the path $\tau(s)$ can be denoted as:

$$C_{\tau(s)} = \{g(p) \,|\, p \in \tau(s)\} \tag{5}$$

To check the decision consistency, the similarity between the covered grids of $\tau(s^*)$ and $\tau(s')$ is computed. The consistency checking is defined using a predefined threshold ε as follows:

$$\epsilon(\tau(s^*), \tau(s')) = \frac{|C_{\tau(s^*)} \cap C_{\tau(s')}|}{|C_{\tau(s^*)} \cup C_{\tau(s')}|} > \varepsilon \tag{6}$$

If the similarity between the covered grids is greater than the threshold ε, s' is recognized as an ODS.

C. Feedback $\mathcal{D}$

To guide the search of NoDSs, a feedback is necessary to select high-quality individuals from the candidate population (Line 15 of Algorithm 1). The grid similarity $\epsilon(\tau(s^*), \tau(s'))$ (in Equation 6) provides a direct choice. However, the calculation of grid similarity is relatively coarse-grained as it only considers the spatial perspective of the ego vehicle's motion. While this coarse-grained calculation is beneficial for consistency checking, it may not be as specific and effective for guiding the testing to generate NoDSs (see our evaluation results in Section V-B2). To mitigate this, another metric is proposed that incorporates more fine-grained feedback calculated from scenario observations (detailed in Section II-B), taking into account the ego vehicle's driving path and motion behavior. The former focuses on the spatial characteristics of the ego vehicle's motion, while the latter more on the temporal ones.

Driving Path Feedback. The main objective of *Decictor* is to maximize the difference between the driving paths such that the non-optimal decision is identified. Considering the potential different lengths of $\tau(s^*)$ and $\tau(s')$, we use the average point-wise distance between $\tau(s^*)$ and $\tau(s')$ in the spatial space to measure their difference, which is calculated as:

$$f_p(s^*, s') = \frac{1}{n_{s'}} \sum_{i=1}^{n_{s'}} \min\{\|p_i - p\| \,|\, \forall p \in \tau(s^*)\} \tag{7}$$

where $p_i \in \tau(s')$ and $n_{s'}$ is the total number of points in $\tau(s')$.

Behavior Feedback. In some cases, optimizing only the driving path feedback is difficult, thus an additional feedback mechanism is provided, which is based on the fine-grained behavior of the ego vehicle. This behavior feedback considers factors such as the ego's velocity, acceleration, and heading. By analyzing the ego's behavior, it becomes possible to gain insights into how slight changes in behavior could lead to variations in the driving path and increase the overall difference between $\tau(s^*)$ and $\tau(s')$. For example, if the heading of the ego vehicle changes slightly, it may not directly cause a significant increase in driving path differences. However, such a change could serve as a valuable indicator, as altering the heading or

acceleration of the ego vehicle could lead to adjustments in the driving path, potentially resulting in an increased difference between paths.

To implement this behavior feedback, the ego's behavior is collected from its waypoints in the scenario observation $O(s)$, denoted as $\mathbf{X}_s = ((\theta_0, v_0, a_0), (\theta_1, v_1, a_1), \ldots, (\theta_k, v_k, a_k))$, where θ_i, v_i and a_i represent the heading, velocity and acceleration at timestamp i, respectively. To compare the behavior differences between two scenarios s^* and s', the Maximum Mean Discrepancy (MMD) is used as the measure of distance between their behavior distributions. The MMD is a widely used statistical metric for comparing distributions and can effectively quantify differences between two sets of data. The behavior feedback function $f_b(s^*, s')$ is defined as:

$$f_b(s^*, s') = f_{MMD}(\mathbf{X}_{s^*}, \mathbf{X}_{s'}). \tag{8}$$

Finally, our fitness score is calculated as:

$$\mathcal{D}(\tau(s^*), \tau(s')) = f_p(s^*, s') + f_b(s^*, s'). \tag{9}$$

V. EMPIRICAL EVALUATION

In this section, we aim to empirically evaluate the capability of *Decictor* on NoDS generation. In particular, we will answer the following research questions:

RQ1: Can *Decictor* effectively find NoDSs for ADSs in comparison to the baselines?

RQ2: How useful are the Non-invasive Mutation and the Feedback designed in *Decictor*?

RQ3: How does *Decictor* perform from the perspective of time efficiency?

To answer these research questions, we conduct experiments using the following settings:

Environment. We implement *Decictor* on Baidu Apollo 7.0 [22] and its built-in simulation environment SimControl. Baidu Apollo 7.0 is an open-source and industrial-level ADS that supports a wide variety of driving supports.

Driving Scenarios. We evaluate *Decictor* on a real-world Map Sunnyvale Loop, provided as part of Baidu Apollo. Following the existing works [8], [9], we use six representative scenarios and build the initial ODSs, the inputs of Algorithm 1. These scenarios have been widely tested in previous literature [8]–[10], [13]–[15], [17], [28], [36] and have covered all possible road types in the Apollo map library. The selected ODSs include left turn (*S1*), right turn (*S2*), lane following (*S3*), U-turn (*S4*), crossing intersection (*S5*), and existing driving road (*S6*). The optimality of seed ODSs is ensured through manual verification.

Baselines. Since this is the first work to specifically evaluate the optimality of PPDs, and there are no baselines that are directly related to PPD testing. Thus, we design two random strategies for fair comparison with our *Decictor*, i.e., Random and Random-δ. Specifically, 1) Random, which does not have any feedback and mutation constraints. It randomly generates scenarios by adding or removing dynamic vehicles or static obstacles; 2) Random-δ, which does not have feedback but uses our non-invasive mutation δ. Besides, we selected seven safety-related baselines for comparison: AVFuzzer [8], SAMOTA [10], BehAVExplor [9], DriveFuzz [13], scenoRITA [14], DoppelTest [15] and DeepCollision [17]. We understand that comparisons with the seven baselines, primarily designed for identifying safety-critical violations, may not be absolutely fair. However, our empirical results demonstrate that they can still generate scenarios with non-optimal PPDs, which would be overlooked without a consistency check. Thus, the comparison with these tools still underscore the need to test the optimality of PPDs.

Note that not all of baselines were originally evaluated in the same simulation environment as *Decictor* (i.e., SimControl + Apollo). Specifically, AVFuzzer, BehAVExplor, and DeepCollision were implemented based on the LGSVL simulator [37], which has been sunsetted [38]. SAMOTA and DriveFuzz are developed for Pylot [27] and Autoware [26], respectively, both running with the CARLA simulator [39]. Therefore, we invested significant effort in migrating them to our simulation environment for comparison. Since their core algorithms (e.g., surrogate models, seed selection, and feedback) are often general, we kept them the same as the original implementation. We mainly modified the simulation interface (changing the simulator APIs) so that they can run on SimControl and Apollo. Furthermore, we set the ego task consistent with our approach to facilitate comparison. Details are available on [40].

Metrics. To facilitate comparison with the baseline techniques, we have incorporated our consistency check into them to gather the generated NoDSs. In our experiments, we utilize the metrics #NoDS and #NoDS-Hum to assess the effectiveness of NoDS generation. #NoDS quantifies the number of potential NoDSs, identified through our consistency checking with a predetermined threshold ε.

As discussed in Section IV-A, although our mutation strategy is conservative, it is still impossible to guarantee with absolute certainty that mutations will not influence ego behavior. Additionally, the ADS may plan an alternative route that, from a human perspective, could also be deemed an optimal decision. This bears resemblance to the generation of invalid inputs in existing adversarial attacks [21] or deep learning testing methodologies [41], despite there are some mutation constraints. Considering these factors, we introduce #NoDS-Hum, representing the number of NoDSs validated by human evaluation. Specifically, the co-authors verify whether the new paths in NoDSs are truly less optimal than the paths in the seed ODSs. A NoDS is only counted in #NoDS-Hum if *all* authors unanimously recognize it as non-optimal, indicating a clear-cut case of a non-optimal decision.

Moreover, we also assess the success rate of non-invasive mutations, denoted as %Mutation. Specifically, the evaluation involves replaying the ego vehicle in the mutated scenario alongside the optimal path of the ODS and verifying whether the motion task can be completed on this path.

Implementation. Following AVFuzzer [8], we set the population size N in *Decictor* to 4. In our mutation, we consider both dynamic vehicles and static obstacles (i.e., traffic cones). According to our preliminary study [42], we set the time step Δt in the non-invasive mutation to 2 seconds and the equality

TABLE I. COMPARISON RESULTS WITH BASELINES.

Method	#NoDS (#NoDS-Hum)↑							%Mutation↑						
	S1	*S2*	*S3*	*S4*	*S5*	*S6*	*Sum*	*S1*	*S2*	*S3*	*S4*	*S5*	*S6*	*Avg.*
AVFuzzer	0.6 (0.3)	1.5 (1.0)	0.0 (0.0)	1.2 (0.5)	1.4 (0.9)	5.0 (0.6)	9.7 (3.3)	41.5	40.5	70.6	10.6	6.7	77.5	41.2
SAMOTA	1.5 (0.9)	0.4 (0.3)	0.0 (0.0)	3.3 (1.2)	7.2 (6.2)	4.0 (0.8)	16.4 (9.4)	58.6	50.1	49.6	55.7	54.1	68.7	56.1
BehAVExplor	2.5 (1.7)	1.6 (0.7)	0.0 (0.0)	6.3 (2.3)	3.8 (3.3)	3.0 (0.4)	17.2 (8.4)	29.7	36.3	36.6	32.1	16.4	40.8	32.0
DriveFuzz	0.1 (0.0)	5.8 (2.6)	0.0 (0.0)	4.2 (1.2)	1.5 (0.9)	4.0 (0.5)	15.6 (5.2)	23.8	55.5	41.2	34.7	17.1	58.5	38.5
scenoRITA	0.7 (0.3)	0.0 (0.0)	0.0 (0.0)	0.9 (0.2)	5.2 (2.7)	2.2 (0.4)	9.0 (3.6)	92.4	94.7	98.2	94.7	96.1	96.9	95.5
DoppelTest	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	1.8 (0.6)	0.0 (0.0)	0.0 (0.0)	1.8 (0.6)	94.7	79.1	97.9	97.6	96.1	97.6	93.8
DeepCollision	2.0 (1.0)	6.1 (1.8)	0.0 (0.0)	3.2 (1.0)	2.1 (0.8)	1.9 (0.2)	15.3 (4.8)	58.0	65.5	44.0	48.4	54.9	62.0	55.5
Random	0.3 (0.0)	3.0 (1.5)	0.2 (0.2)	1.9 (0.7)	7.6 (6.8)	3.7 (0.3)	16.7 (9.5)	73.2	73.1	70.2	76.7	65.3	66.0	70.8
Random-δ	3.0 (1.4)	5.1 (4.6)	4.4 (4.4)	7.2 (2.0)	10.5 (8.1)	5.2 (0.4)	35.4 (20.9)	96.4	97.4	99.0	96.0	95.6	98.1	97.1
Decictor	**9.4 (3.9)**	**11.9 (8.6)**	**9.0 (9.0)**	**15.7 (6.7)**	**12.1 (9.4)**	**5.8 (1.1)**	**63.9 (38.7)**	**98.0**	**97.5**	**99.4**	**97.5**	**96.6**	**98.2**	**97.9**

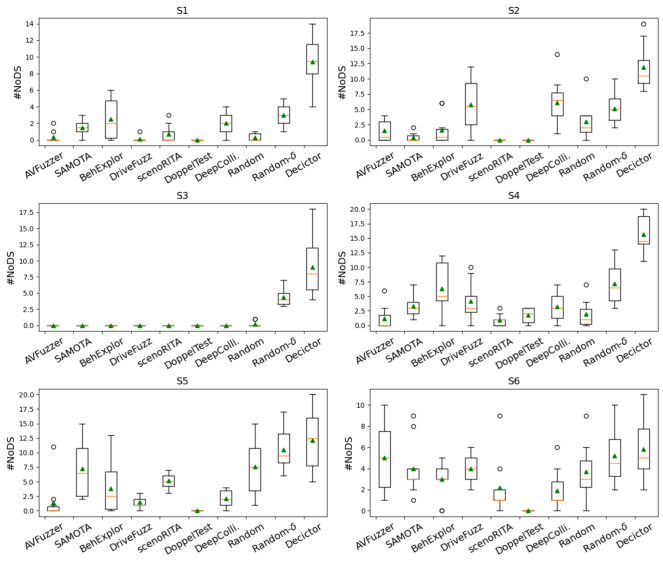


Fig. 5. Comparison of #NoDS for different tools.

threshold in the Consistency Check to $\varepsilon = 0.6$. The grid size is set to 2 meters to recognize when the entire vehicle deviates from the optimal path. Similar to previous works [10], [43], we repeat each experiment 10 times to mitigate the influence of the non-determinism inherent in the simulation-based execution of the ADS. For each run, the budget is set as four hours.

A. RQ1: Effectiveness of Decictor

1) Comparative Results: Table I compares #NoDS, #NoDS-Hum, and %Mutation across six initial ODSs: *S1*, *S2*, *S3*, *S4*, *S5* and *S6*, and Fig. 5 illustrates the distribution of #NoDS across ten iterations of each method in every scenario, where the median and the average are represented by an orange bar and a green triangle, respectively. From the results, we can find that *Decictor* outperforms the baselines in #NoDS and #NoDS-Hum. In detail, on average of #NoDS, *Decictor* outperforms the best baseline (i.e., Random-δ): *S1* (9.4 vs. 3.0), *S2* (11.9 vs. 5.1), *S3* (9.0 vs. 4.4), *S4* (15.7 vs. 7.2), *S5* (12.1 vs. 10.5) and *S6* (5.8 vs. 5.2). Among all the detected NoDSs by *Decictor*, we finally identified 3.9, 8.6, 9.0, 6.7, 9.4 and 1.1 NoDS-Hum in *S1*, *S2*, *S3*, *S4*, *S5* and *S6* respectively. The detailed human verification results for NoDS-Hum are given on our website [40]. Even with very rigorous manual filtering, *Decictor* still significantly outperforms the best baseline (Random-δ).

The results indicate the effectiveness of *Decictor* in identifying NoDSs and NoDS-Hum. We further observed that Random-δ outperforms both the Random (35.4 vs. 16.7 for the total number of NoDSs) and the best safety-oriented baseline BehAVExplor (35.4 vs. 17.2 in *Sum* of #NoDS). This is attributed to the effectiveness of the non-invasive mutation used in Random-δ and *Decictor*. It is worth noting that BehAVExplor performs better than the other safety-guided baselines due to its diversity feedback mechanism (with diverse explorations), which makes it more likely to generate NoDSs. However, scenoRITA and DoppelTest exhibit lower performance. This occurs because they search for mutations in a wider road area than others, resulting in many mutations that do not impact the ego vehicle's motion. scenoRITA outperforms DoppelTest as it incorporates a larger number of obstacles in its mutation processes, increasing the probability of generating NoDSs. We also observed that all safety-oriented baselines fail to perform on S3. This is primarily due to their lack of consideration for the impact of static obstacles during mutation, which is the key reason for NoDSs in S3. #NoDS-Hum is lower than #NoDS due to the very strict criteria set for determining optimality.

Regarding %Mutation (*Avg.* column), Table I reveals that Random and safety-oriented baselines produce a maximum of 95.5% valid mutations (i.e., the original optimal path is not affected), which is lower than non-invasive mutation methods (97.1% for Random-δ and 97.9% for *Decictor* on average). The high rate of valid mutations demonstrates that our non-invasive mutation effectively ensures the validity of mutated objects. Note that compared with other safety-oriented baselines, scenoRITA and DoppelTest generate a higher number of valid mutations. This is because they explore a broader mutation space, most of which is disjoint from the motion space of the ego vehicle, which is less likely to conflict with the optimal path of the ego vehicle. Compared to Random-δ, the average 0.8% increase of *Decictor* in %Mutation indicates that integrating the feedback mechanism not only maintained the effectiveness of our mutation process but also increased the detection of NoDSs.

In summary, the comparison of *Decictor* with Random and Random-δ demonstrates its effectiveness in detecting NoDSs. When contrasted with the seven state-of-the-art baselines, the results emphasize the importance of robust path-planning decision testing. *Decictor* and these safety-oriented baselines are complementary, given their distinct testing objectives.

2) Root Causes of NoDSs: To better understand the detected NoDSs, we manually summarize six patterns based on the potential root causes for these non-optimal decisions. Table II

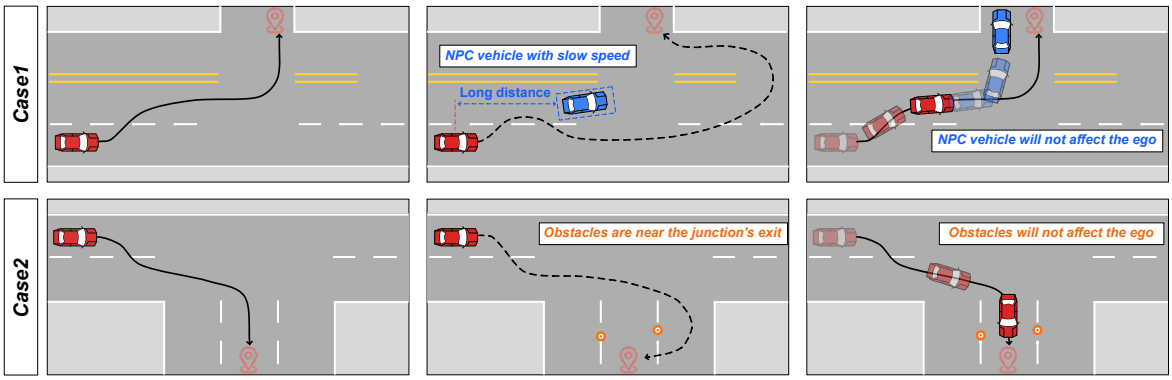


Fig. 6. Cases of NoDSs detected by Decictor. The red vehicle is the AV. Optimal paths are indicated **Solid lines**, while non-optimal paths are displayed in **Dotted lines**. The first column is the initial ODSs, the second column shows the corresponding NoDSs, and the last column depicts the reproduced scenarios validating that the original optimal paths can be traversed in the NoDSs.

TABLE II. PROPORTION OF DISCOVERED NODSS

No.	Root Cause	Prop.(%)	Uni.	Uni.*
1	Inaccurate prediction of vehicle status	6	✓	✓
2	Inaccurate prediction of obstacle impact	29	✓	✗
3	Inaccurate estimation of accessible region	28	✓	✗
4	Inaccurate prediction of other vehicles' intentions	15	✓	✓
5	Excessive attention to surrounding obstacles	10	✓	✗
6	Excessively cautious about nearby vehicles	12	✗	✗

shows the six unique root causes:

1) The ADS inaccurately predicts a moving NPC vehicle as stationary, leading to re-plan a non-optimal path.
2) The ADS inaccurately enlarge the impact region of static obstacles on the ego vehicle's lane boundaries, resulting in the change of the optimal path.
3) The ADS inaccurately estimates accessible regions between parallel obstacles that do not obstruct the optimal path.
4) The ADS overly prioritizes an unrelated NPC's movements, reducing feasible space for optimal path decision.
5) The ADS overly focuses on keeping a safe distance from surrounding static obstacles, resulting in insufficient space to select the optimal path.
6) The ADS is overly cautious about trailing vehicles, misclassifying part of the optimal area as non-optimal, resulting in insufficient space for executing the optimal path.

Table II shows the distribution of NoDSs discovered by *Decictor*, with the respective proportions for the six root causes being 6%, 29%, 28%, 15%, 10%, and 12%, demonstrating *Decictor*'s capability to identify a diverse range of root causes in NoDSs. The Uni. column underscores that *Decictor* identifies five unique root causes (No. 1 to 5) compared to the seven existing safety-critical ADS testing baselines. Notably, our method detects two unique NoDSs (No. 1 and 4) not identified by the Random-based baseline, *Random-δ* and *Random*, thus highlighting *Decictor*'s enhanced ability to discover varied root causes in NoDSs.

3) Case Study: Fig. 6 shows two representative NoDS examples. More NoDS examples and corresponding videos can be found on our website [40]. The examples illustrate that the ego vehicle takes non-optimal paths in the detected NoDSs, even though the optimal ones are still available.

Case 1. Inaccurate Prediction of Vehicle Status. The *Decictor* adds a new NPC vehicle in the NoDS, initially slow and located at a distance to the ego vehicle. As shown in the third column of Fig. 6, by the time the ego vehicle approaches, the NPC has moved, yet the original optimal path is still available. However, due to an incorrect prediction treating the NPC vehicle as stationary, the ADS stops lane changing and chooses a less efficient path.

Case 2. Inaccurate Prediction of Obstacle Impact. *Decictor* places two obstacles near the junction exit, not affecting the optimal path. However, the ego vehicle (i.e., 2.1-meter width) deems the target lane (i.e., 3.5-meter width) impassable, choosing a suboptimal path from an adjacent lane, which raises risks in real-world traffic scenarios.

Answer to RQ1: *Decictor* significantly outperforms existing methods in identifying NoDSs. The non-invasive mutation can accurately generate valid scenarios (96.6% - 99.4%).

B. RQ2: Usefulness of Mutation and Feedback

We evaluated the key components of *Decictor*, including the *non-invasive mutation* and *feedback* mechanism, by configuring a series of *Decictor* variants.

1) Mutation: For mutation, we compared *Decictor* with its three variants: (1) *w/o Cons* applies a random mutation instead of the non-invasive mutation in *Decictor*, aiming to evaluate the effectiveness of the non-invasive mutation; (2) *w/o Mot* sets the time step to 0s in the calculation of non-invasive feasible areas (i.e., only considering the motion constraint at each timestamp), aiming to measure the influence of the motion constraint between two successive timestamps. (3) *w/o Rem* uses only the adding operation in *Decictor* under the constraint of non-invasive mutation, aiming to assess the effectiveness of the combination of the two mutation operations, i.e., adding and removing participants. As shown in Table III, we find that *w/o Cons* generates the fewest valid mutations (69.2%) and detects the smallest number of NoDSs (25.8), underlying the importance of the non-invasive mutation. Comparing *w/o Mot* with *Decictor*, we observed that only considering the motion constraint at each timestamp is insufficient, and the continuous-time motion constraint plays a significant role in calculating the non-invasive feasible area. Our preliminary study revealed that %Mutation increases while the #NoDS decreases as the time step increases. The reason is that a long time step will yield

TABLE III. EFFECTIVENESS OF MUTATIONS

Metric	Method	*S1*	*S2*	*S3*	*S4*	*S5*	*S6*	*Avg.*	*Sum*
%Mutation ↑	**w/o Cons**	71.7	71.3	74.8	65.5	67.0	64.7	69.2	-
	w/o Mot	71.1	80.9	98.6	68.9	77.6	66.1	77.2	-
	w/o Rem	97.5	97.3	99.4	97.2	91.8	94.0	96.2	-
	Decictor	**98.0**	**97.5**	**99.4**	**97.5**	**96.6**	**98.2**	**97.9**	-
#NoDS ↑	**w/o Cons**	2.7	4.5	1.6	5.7	7.0	4.3	-	25.8
	w/o Mot	4.0	8.6	5.2	10.8	11.7	5.0	-	45.3
	w/o Rem	3.9	9.3	4.9	9.1	8.3	5.7	-	41.2
	Decictor	**9.4**	**11.9**	**9.0**	**15.7**	**12.1**	**5.8**	-	**63.9**

TABLE IV. COMPARISON OF FEEDBACK

Method	#NoDS ↑						
	S1	*S2*	*S3*	*S4*	*S5*	*S6*	*Sum*
F-Random	3.0	5.1	4.4	7.2	10.5	5.2	35.4
F-Con	3.0	6.7	1.2	7.5	11.7	4.9	35.0
F-Path	4.3	10.8	5.0	10.6	12.1	5.4	48.2
F-Behavior	5.3	9.0	5.4	11.5	11.8	5.6	48.6
Decictor	**9.4**	**11.9**	**9.0**	**15.7**	**12.1**	**5.8**	**63.9**

scenarios with reduced interactivity between the ego vehicle and the added participant, thus exerting a lower impact on the PPD process of the ADS. Comparing *w/o Rem* and *Decictor*, we found that mutation with only the adding operation is more likely to induce more invalid mutations and a lower number of NoDSs. This is because the adding operation introduces too many obstacles, potentially leading to the failure of the motion task. Thus, the removing operation is also important for generating NoDSs.

2) Feedback: For feedback, we implemented four variants: (1) *F-Random* replaces the feedback of *Decictor* with a random selection (from $\mathbf{Q} \cup \mathbf{Q}'$) to evaluate the effectiveness of our feedback strategy. (2) *F-Con* replaces the feedback of *Decictor* with the grid similarity used in our consistency checking (see Equation 6) to compare the performance between using a coarse-grained grid distance and a fine-grained path distance. (3) *F-Path* and (4) *F-Behavior* consider only the *driving path feedback* and the *behavior feedback*, respectively, to evaluate the usefulness of either feedback type.

Table IV shows the experimental results of #NoDSs. The comparative results between *F-Random* and *Decictor* (35.4 vs. 63.9 in total) illustrate the effectiveness of the feedback used in *Decictor*. The results of *F-Con* (35.0) indicates that the grid similarity is not an effective feedback metric, as it may overlook some scenarios with a high probability of inducing NoDSs. From the ablation results of *F-Path* and *F-Behavior* (48.2 and 48.6, respectively), we see that both *behavior feedback* and *driving path feedback* are beneficial for detecting NoDSs. Their combination achieves the best performance.

Note that, we also evaluated the influence of different Δt used in non-invasive mutation. Due to the space limit, the detailed experimental results can be found on our website [40].

Answer to RQ2: Both the non-invasive mutation and the two types of feedback are useful for *Decictor* to detect non-optimal decisions.

C. RQ3: Test Efficiency of Decictor

Fig. 7 shows the cumulative number of NoDSs over the execution time of different methods. We can find that as *Decictor* continues its execution, the detection of NoDSs steadily increases, whereas other methods quickly reach a stable state. We further assess the time performance of different components in *Decictor*, including the overhead of mutation, oracle checking, feedback calculation, and simulation. Specifically, we analyze the average time to handle a scenario. The results are summarized in Table V, where the columns *Mutation*, *Oracle Check*, *Feedback*, and *Simulation* represent the average time taken by a scenario for mutation, validity checking, fitness computation, and running the scenario in the simulation platform, respectively. 0.01* represents that the time cost is small, i.e., less than 0.01. For Random, the oracle and feedback stages are not involved, being marked as 'N/A'.

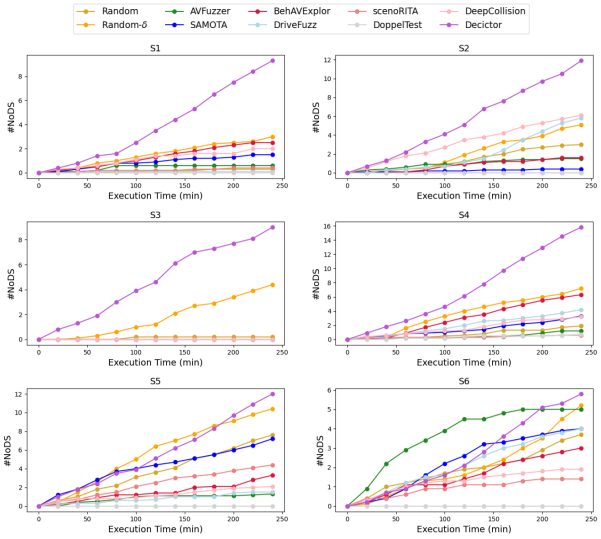


Fig. 7. #NoDSs over the execution of different methods.

TABLE V. RESULTS OF TIME PERFORMANCE (S)

Method	Mutation	Oracle Check	Feedback	Simulation	Total
Random	2.28	N/A	N/A	66.67	68.95
Random-δ	8.04	N/A	N/A	65.30	73.34
AVFuzzer	1.46	0.01*	0.01*	71.26	72.74
SAMOTA	1.18	0.01*	32.11	65.05	98.35
BehAVExplor	0.56	0.01*	1.41	77.06	79.04
DriveFuzz	2.02	0.01*	0.01*	77.56	79.60
scenoRITA	0.55	0.01*	22.16	72.70	95.42
DoppelTest	0.01	0.01*	21.36	69.06	90.44
DeepCollision	0.19	0.01*	0.02	66.42	66.64
Decictor	8.03	1.62	0.73	67.20	77.58

The overall results show that, for all tools, the simulation process spends the majority of time. For instance, on average, *Decictor* takes 77.58 seconds to process a scenario, where 67.20 seconds (86.62%) are used in running the scenario. *Decictor* spends slightly more time in Mutation and Oracle Check than others due to the computation of the non-invasive feasible area for each waypoint and the grid-based similarity-checking mechanism. Note that we only consider the mutation time when generating scenario configurations. However, the computation time of *Decictor* remains within an acceptable range. SAMOTA consumes the most time in the feedback due to the necessity of training a surrogate model. The feedback times for scenoRITA and DoppelTest are approximately twice as long as those reported in [15]. This is due to the simulation length was set to twice the original settings to ensure the

ego could complete the task. The differing time duration for simulation across various tools can be attributed to the different scenarios each tool generates (e.g., time-out scenarios), which in turn require varying amounts of simulation time.

Answer to RQ3: *Decictor* demonstrates efficiency, with the majority of the time (86.62%) spent in the simulation phase, while the main algorithmic component consumes approximately 10.38 seconds (13.38%).

D. Threats to Validity

Decictor suffers from some threats. First, the selection of the seed ODSs could potentially influence the results. To mitigate this, we have judiciously selected six motion tasks and created corresponding scenarios. The decision optimality of each ODS was manually analyzed and confirmed by all authors, including a co-author with industrial expertise. Second, the threshold selection for the Consistency Check ϵ in *Decictor* presents another threat, as different thresholds can lead to varying interpretations of non-optimal path decisions. To counter this, on one hand, we carefully select the threshold ε in the Consistency Check and Δt in the computation of non-invasive feasible area (the empirical evaluation of the two parameters can be found on our website [40]) to reduce the number of such scenarios; on the other hand, to ensure robustness and consistency, all co-authors independently review the generated NoDSs, and only those confirmed by unanimous agreement among all authors will be considered in NoDS-Hum. Third, the non-determinism inherent in ADS execution may also pose a threat to our results, and we mitigate it by repeating each experiment multiple times.

Lastly, the environments used could pose a potential threat. On one hand, we adapted existing baselines to the Apollo+SimControl simulation environment, which could potentially influence the results. However, our modifications were mainly limited to the interface between the algorithms and the simulation environment, leaving the core algorithms unchanged. We thoroughly reviewed the code and released the implementation of baselines on our website [40].

VI. Related Work

Safety-Guided ADS Testing. Safety plays the most important role in the development of ADSs. However, guaranteeing everlasting safety is challenging and many approaches have been proposed to generate safety-critical scenarios for evaluating the safety of ADSs. They can be divided into data-driven approaches [3], [4], [6], [44]–[48] and searching-based approaches [7]–[11], [15], [33], [36], [49]–[53]. Data-driving methods produce critical scenarios from real-world data, such as traffic recordings [4], [6], [47], [54] and accident reports [3], [44]–[46], [55], [56]. Search-based methods use various technologies to search for safety-critical scenarios from the scenario space, such as guided fuzzing [8], [9], [57], [58], evolutionary algorithms [7], [16], [33], [36], [50]–[52], metamorphic testing [59], surrogate models [10], [11], reinforcement learning [17], [43], [60] and reachability analysis [49], [61]. All these methods aim to generate critical scenarios for evaluating ADSs' safety-critical requirements, such as collision avoidance and reaching the destination. In contrast, our work evaluates the non-safety-critical requirements of ADSs, which is also crucial for the real-world deployment of ADSs. Compared to these works, the key difference is that *Decictor* is the first to evaluate non-safety-critical requirements for ADSs, particularly in the aspect of non-optimal path-planning decisions, whereas the other works mainly focus on safety-critical issues.

Robustness Analysis for ADS. Robustness is a crucial issue for autonomous driving systems (ADSs). Recent studies have shown that the perception module in ADSs (i.e., object detection) exhibit vulnerabilities to various threats including perturbing sensor signals [62], [63], modifying objects [64]–[66], etc. There are also some studies [67], [68] work on the robustness of the prediction module by crafting trajectories of other vehicles. However, no existing work has tested the path-planning robustness of autonomous driving systems. In this paper, we propose the first work to reveal non-optimal PPDs of the ADS, which can be meaningful to inspiring the improvement on ADS robustness and reliability.

Path-planning in ADSs. Obtaining optimal PPDs is a challenging task in ADSs [69]. Existing studies on path planning primarily focus on avoiding obstacles, overlooking other attributes that contribute to optimality (e.g., motion time, path length, and comfort). Traditional algorithms [70]–[73] often result in discontinuous paths, leading to non-optimal overall decisions. While, machine learning-based planning algorithms [74]–[76] could become trapped in local minima, unable to find optimal solutions. Therefore, given an ADS, it is critical to evaluate the robustness of its optimal PPDs, but there is still a significant gap in this area. To mitigate this, we proposed the first PPD testing technique.

VII. Conclusion

In this paper, we propose the first study to evaluate the robustness of ADSs' path-planning decisions. To address this issue, we develop a testing tool *Decictor*, which comprises three main components: Non-invasive Mutation, generating new scenarios that are unlikely to affect the original optimal paths; Consistency Check, determining whether the driving paths in the mutated scenarios are consistent with the original optimal one; Feedback-guided Selection, responsible for selecting better scenarios for the subsequent mutation. The experimental results demonstrate the effectiveness and efficiency of *Decictor*, as well as the usefulness of each component in *Decictor*.

Acknowledgment

This research is partially supported by the Ministry of Education, Singapore under its Academic Research Fund Tier 2 (Proposal ID: T2EP20223-0043), the Lee Kong Chian Fellowship, the National Natural Science Foundation of China (62102283) and the Beijing Nova Program. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

REFERENCES

[1] J. Garcia, Y. Feng, J. Shen, S. Almanee, Y. Xia, and Q. A. Chen, "A comprehensive study of autonomous vehicle bugs," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. Seoul, South Korea: IEEE, 2020, pp. 385–396.

[2] G. Lou, Y. Deng, X. Zheng, M. Zhang, and T. Zhang, "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*, 2022, pp. 31–43.

[3] X. Zhang and Y. Cai, "Building critical testing scenarios for autonomous driving from real accidents," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 462–474.

[4] Y. Deng, X. Zheng, M. Zhang, G. Lou, and T. Zhang, "Scenario-based test reduction and prioritization for multi-module autonomous driving systems," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 82–93.

[5] S. K. Bashetty, H. B. Amor, and G. Fainekos, "DeepCrashTest: Turning dashcam videos into virtual crash tests for automated driving systems," in *2020 IEEE International Conference on Robotics and Automation ICRA*. Paris, France: IEEE, 2020, pp. 11 353–11 360.

[6] J.-P. Paardekooper, S. Montfort, J. Manders, J. Goos, E. d. Gelder, O. Camp, O. Bracquemond, and G. Thiolon, "Automatic identification of critical scenarios in a public dataset of 6000 km of public-road driving," in *26th International Technical Conference on the Enhanced Safety of Vehicles (ESV)*. Eindhoven, Netherlands: Mira Smart, 2019.

[7] A. Gambi, M. Mueller, and G. Fraser, "Automatically testing self-driving cars with search-based procedural content generation," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. Beijing, China: ACM, 2019, pp. 318–328.

[8] G. Li, Y. Li, S. Jha, T. Tsai, M. Sullivan, S. K. S. Hari, Z. Kalbarczyk, and R. Iyer, "AV-FUZZER: Finding safety violations in autonomous driving systems," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. Coimbra, Portugal: IEEE, 2020, pp. 25–36.

[9] M. Cheng, Y. Zhou, and X. Xie, "Behavexplor: Behavior diversity guided testing for autonomous driving systems," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 488–500.

[10] F. U. Haq, D. Shin, and L. Briand, "Efficient online testing for dnn-enabled systems using surrogate-assisted and many-objective optimization," in *Proceedings of the 44th International Conference on Software Engineering*. Pittsburgh Pennsylvania: IEEE, 2022, pp. 811–822.

[11] Z. Zhong, G. Kaiser, and B. Ray, "Neural network guided evolutionary fuzzing for finding traffic violations of autonomous vehicles," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 1860–1875, 2023.

[12] A. Aleti *et al.*, "Identifying safety-critical scenarios for autonomous vehicles via key features," *arXiv preprint arXiv:2212.07566*, 2022.

[13] S. Kim, M. Liu, J. J. Rhee, Y. Jeon, Y. Kwon, and C. H. Kim, "Drivefuzz: Discovering autonomous driving bugs through driving quality-guided fuzzing," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. Los Angeles, CA, USA: ACM, 2022, pp. 1753–1767.

[14] Y. Huai, S. Almanee, Y. Chen, X. Wu, Q. A. Chen, and J. Garcia, "sceno rita: Generating diverse, fully-mutable, test scenarios for autonomous vehicle planning," *IEEE Transactions on Software Engineering*, 2023.

[15] Y. Huai, Y. Chen, S. Almanee, T. Ngo, X. Liao, Z. Wan, Q. A. Chen, and J. Garcia, "Doppelgänger test generation for revealing bugs in autonomous driving software," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 2591–2603.

[16] H. Tian, Y. Jiang, G. Wu, J. Yan, J. Wei, W. Chen, S. Li, and D. Ye, "Mosat: finding safety violations of autonomous driving systems using multi-objective genetic algorithm," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 94–106.

[17] C. Lu, Y. Shi, H. Zhang, M. Zhang, T. Wang, T. Yue, and S. Ali, "Learning configurations of operating environment of autonomous vehicles to maximize their collisions," *IEEE Transactions on Software Engineering*, vol. 49, no. 1, pp. 384–402, 2022.

[18] S. P. Force, "FINAL theory of driving," https://www.police.gov.sg/-/media/Spf/Files/TP/Online-Learning-Portal/FT-ENG-9th-Edition-130717.ashx, 2017.

[19] AI DRIVR, "The state of Tesla FSD," 2023. [Online]. Available: https://www.youtube.com/watch?v=2VWyaAzwMT0&ab_channel=AIDRIVR

[20] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6572

[21] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *IEEE Symposium on Security and Privacy*, May 2017, pp. 39–57.

[22] Baidu, "Apollo: Open source autonomous driving," 2019. [Online]. Available: https://github.com/ApolloAuto/apollo

[23] Z. Zhang, A. Liniger, D. Dai, F. Yu, and L. Van Gool, "End-to-end urban driving by imitating a reinforcement learning coach," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 15 222–15 232.

[24] Y. Hu, J. Yang, L. Chen, K. Li, C. Sima, X. Zhu, S. Chai, S. Du, T. Lin, W. Wang, L. Lu, X. Jia, Q. Liu, J. Dai, Y. Qiao, and H. Li, "Planning-oriented autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[25] comma.ai, "OpenPilot: An open source driver assistance system," 2022. [Online]. Available: https://github.com/commaai/openpilot

[26] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018, pp. 287–296.

[27] I. Gog, S. Kalra, P. Schafhalter, M. A. Wright, J. E. Gonzalez, and I. Stoica, "Pylot: A modular platform for exploring latency-accuracy tradeoffs in autonomous vehicles," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 8806–8813.

[28] E. Thorn, S. C. Kimmel, M. Chaka, B. A. Hamilton *et al.*, "A framework for automated driving system testable cases and scenarios," United States. Department of Transportation. National Highway Traffic Safety, Tech. Rep., 2018.

[29] K. Leino, Z. Wang, and M. Fredrikson, "Globally-robust neural networks," in *International Conference on Machine Learning*. PMLR, 2021, pp. 6212–6222.

[30] W. Ruan, M. Wu, Y. Sun, X. Huang, D. Kroening, and M. Kwiatkowska, "Global robustness evaluation of deep neural networks with provable guarantees for the hamming distance," *IJCAI-19*, 2019.

[31] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE symposium on security and privacy (SP)*. IEEE, 2016, pp. 582–597.

[32] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*. Ieee, 2017, pp. 39–57.

[33] Y. Tang, Y. Zhou, T. Zhang, F. Wu, Y. Liu, and G. Wang, "Systematic testing of autonomous driving systems using map topology-based scenario classification," in *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Melbourne, Australia: IEEE, 2021, pp. 1342–1346.

[34] L. Wang, X. Xie, X. Du, M. Tian, Q. Guo, Z. Yang, and C. Shen, "Distxplore: Distribution-guided testing for evaluating and enhancing deep learning systems," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 68–80.

[35] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *2015 IEEE intelligent vehicles symposium (IV)*. IEEE, 2015, pp. 1094–1099.

[36] Y. Zhou, Y. Sun, Y. Tang, Y. Chen, J. Sun, C. M. Poskitt, Y. Liu, and Z. Yang, "Specification-based autonomous driving system testing," *IEEE Transactions on Software Engineering*, pp. 1–19, 2023.

[37] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta *et al.*, "LGSVL simulator: A high fidelity simulator for autonomous driving," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. Rhodes, Greece: IEEE, 2020, pp. 1–6.

[38] LG Electronics, "SVL Simulator Sunset," https://www.svlsimulator.com/news/2022-01-20-svl-simulator-sunset/.

[39] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual*

Conference on Robot Learning, ser. Proceedings of Machine Learning Research, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., vol. 78. California, USA: PMLR, 2017, pp. 1–16.

[40] M. Cheng, X. Xie, Y. Zhou, J. Wang, G. Meng, and K. Yang, "Decictor," https://sites.google.com/view/adsdecictor, 2024.

[41] S. Dola, M. B. Dwyer, and M. L. Soffa, "Distribution-aware testing of neural networks using generative models," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 226–237.

[42] M. Cheng, X. Xie, Y. Zhou, J. Wang, G. Meng, and K. Yang, "Decictor Preliminary Study," https://sites.google.com/view/adsdecictor/primiliary-study, 2024.

[43] F. U. Haq, D. Shin, and L. C. Briand, "Many-objective reinforcement learning for online testing of dnn-enabled systems," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1814–1826.

[44] A. Gambi, T. Huynh, and G. Fraser, "Generating effective test cases for self-driving cars from police reports," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Tallinn Estonia: ACM, 2019, pp. 257–267.

[45] W. G. Najm, S. Toma, J. Brewer *et al.*, "Depiction of priority light-vehicle pre-crash scenarios for safety applications based on vehicle-to-vehicle communications," National Highway Traffic Safety Administration, U.S. Department of Transportation, Washington, DC, Tech. Rep. DOT HS 811 732, Apr. 2013.

[46] P. Nitsche, P. Thomas, R. Stuetz, and R. Welsh, "Pre-crash scenarios at road junctions: A clustering method for car crash data," *Accident Analysis & Prevention*, vol. 107, pp. 137–151, 2017.

[47] C. Roesener, F. Fahrenkrog, A. Uhlig, and L. Eckstein, "A scenario-based assessment approach for automated driving by using time series classification of human-driving behaviour," in *2016 IEEE 19th international conference on intelligent transportation systems (ITSC)*. Rio de Janeiro, Brazil: IEEE, 2016, pp. 1360–1365.

[48] Y. Lu, Y. Tian, Y. Bi, B. Chen, and X. Peng, "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*, 2024, pp. 376–388.

[49] C. Hildebrandt, M. von Stein, and S. Elbaum, "Physcov: Physical test coverage for autonomous vehicles," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 449–461.

[50] S. Han, J. Kim, G. Kim, J. Cho, J. Kim, and S. Yoo, "Preliminary evaluation of path-aware crossover operators for search-based test data generation for autonomous driving," in *2021 IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST)*. Madrid, Spain: IEEE, 2021, pp. 44–47.

[51] Y. Tang, Y. Zhou, F. Wu, Y. Liu, J. Sun, W. Huang, and G. Wang, "Route coverage testing for autonomous vehicles via map modeling," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. Xi'an, China: IEEE, 2021, pp. 11 450–11 456.

[52] Y. Tang, Y. Zhou, Y. Liu, J. Sun, and G. Wang, "Collision avoidance testing for autonomous driving systems on complete maps," in *2021 IEEE Intelligent Vehicles Symposium (IV)*. Nagoya, Japan: IEEE, 2021, pp. 179–185.

[53] Z. Li, X. Wu, D. Zhu, M. Cheng, S. Chen, F. Zhang, X. Xie, L. Ma, and J. Zhao, "Generative model-based testing on decision-making policies," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 243–254.

[54] S. Tang, Z. Zhang, J. Zhou, Y. Zhou, Y.-F. Li, and Y. Xue, "Evoscenario: Integrating road structures into critical scenario generation for autonomous driving system testing," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 309–320.

[55] A. Guo, Y. Zhou, H. Tian, C. Fang, Y. Sun, W. Sun, X. Gao, A. T. Luu, Y. Liu, and Z. Chen, "Sovar: Build generalizable scenarios from accident reports for autonomous driving testing," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 268–280.

[56] S. Tang, Z. Zhang, J. Zhou, L. Lei, Y. Zhou, and Y. Xue, "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*, 2024, pp. 1497–1508.

[57] Q. Pang, Y. Yuan, and S. Wang, "Mdpfuzz: Testing models solving markov decision processes," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 378–390. [Online]. Available: https://doi.org/10.1145/3533767.3534388

[58] H. Tian, G. Wu, J. Yan, Y. Jiang, J. Wei, W. Chen, S. Li, and D. Ye, "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*, 2022, pp. 1–12.

[59] J. C. Han and Z. Q. Zhou, "Metamorphic fuzz testing of autonomous vehicles," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020, pp. 380–385.

[60] S. Feng, H. Sun, X. Yan, H. Zhu, Z. Zou, S. Shen, and H. X. Liu, "Dense reinforcement learning for safety validation of autonomous vehicles," *Nature*, vol. 615, no. 7953, pp. 620–627, 2023.

[61] M. Althoff and S. Lutz, "Automatic generation of safety-critical test scenarios for collision avoidance of road vehicles," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. Changshu, Suzhou, China: IEEE, 2018, pp. 1326–1333.

[62] Y. Li, C. Wen, F. Juefei-Xu, and C. Feng, "Fooling lidar perception via adversarial trajectory perturbation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 7898–7907.

[63] Y. Cao, C. Xiao, B. Cyr, Y. Zhou, W. Park, S. Rampazzi, Q. A. Chen, K. Fu, and Z. M. Mao, "Adversarial sensor attack on lidar-based perception in autonomous driving," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 2267–2281.

[64] X. Gao, Z. Wang, Y. Feng, L. Ma, Z. Chen, and B. Xu, "Multitest: Physical-aware object insertion for testing multi-sensor fusion perception systems," *arXiv preprint arXiv:2401.14314*, 2024.

[65] ——, "Benchmarking robustness of ai-enabled multi-sensor fusion systems: Challenges and opportunities," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 871–882.

[66] Y. Cao, S. H. Bhupathiraju, P. Naghavi, T. Sugawara, Z. M. Mao, and S. Rampazzi, "You can't see me: Physical removal attacks on {LiDAR-based} autonomous vehicles driving frameworks," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 2993–3010.

[67] Q. Zhang, S. Hu, J. Sun, Q. A. Chen, and Z. M. Mao, "On adversarial robustness of trajectory prediction for autonomous vehicles," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 15 159–15 168.

[68] Y. Cao, C. Xiao, A. Anandkumar, D. Xu, and M. Pavone, "Advdo: Realistic adversarial attacks for trajectory prediction," in *European Conference on Computer Vision*. Springer, 2022, pp. 36–52.

[69] M. Reda, A. Onsy, A. Y. Haikal, and A. Ghanbari, "Path planning algorithms in the autonomous driving system: A comprehensive review," *Robotics and Autonomous Systems*, vol. 174, p. 104630, 2024.

[70] C. W. Warren, "Fast path planning using modified a* method," in *[1993] Proceedings IEEE International Conference on Robotics and Automation*. IEEE, 1993, pp. 662–667.

[71] M. Thoresen, N. H. Nielsen, K. Mathiassen, and K. Y. Pettersen, "Path planning for ugvs based on traversability hybrid a," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1216–1223, 2021.

[72] S. Spanogiannopoulos, Y. Zweiri, and L. Seneviratne, "Sampling-based non-holonomic path generation for self-driving cars," *Journal of Intelligent & Robotic Systems*, vol. 104, no. 1, p. 14, 2022.

[73] B. Li, Y. Ouyang, L. Li, and Y. Zhang, "Autonomous driving on curvy roads without reliance on frenet frame: A cartesian-based trajectory planning method," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 15 729–15 741, 2022.

[74] Y. Jiang, Z. Liu, D. Qian, H. Zuo, W. He, and J. Wang, "Robust online path planning for autonomous vehicle using sequential quadratic programming," in *2022 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2022, pp. 175–182.

[75] Í. B. Viana and N. Aouf, "Distributed cooperative path-planning for autonomous vehicles integrating human driver trajectories," in *2018 International Conference on Intelligent Systems (IS)*, 2018.

[76] Y. Zhang, H. Sun, J. Zhou, J. Pan, J. Hu, and J. Miao, "Optimal vehicle path planning using quadratic optimization for baidu apollo open platform," in *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2020, pp. 978–984.