Simulation Nondeterminism and Its Impact on Flaky Tests for Autonomous Driving Systems

Olek Osikowicz[1*], Phil McMinn[1] and Donghwan Shin[1]

[1*]School of Computer Science, The University of Sheffield, Western Bank, Sheffield, S10 2TN, South Yorkshire, UK.

*Corresponding author(s). E-mail(s): amosikowicz1@sheffield.ac.uk;
Contributing authors: p.mcminn@sheffield.ac.uk; d.shin@sheffield.ac.uk;

Abstract

Simulation-based testing is a dominant approach for verifying Autonomous Driving Systems (ADS), where test scenarios are executed in virtual environments to evaluate system behavior under challenging conditions. A key assumption behind simulation-based testing is that executions are deterministic, enabling consistent and trustworthy evaluation results. However, recent evidence suggests that this assumption does not consistently hold in practice.

This paper empirically investigates simulation nondeterminism, its underlying causes, and its impact on the flakiness of ADS test results. Our study examines two widely-used open-source driving simulators: CARLA and MetaDrive. Our results show that, unexpectedly, 43.6% of benchmark test scenarios are flaky due to nondeterministic executions in CARLA, whereas MetaDrive exhibits a high level of repeatability across reruns. We further discuss likely causes of nondeterministic simulations and present set of practical strategies for mitigating flakiness in future works.

Keywords: Autonomous Driving Systems, Simulation-based testing, Simulation Nondeterminism, Test Flakiness

1 Introduction

Verifying Autonomous Driving Systems (ADS) against their safety requirements is a crucial step toward widespread deployment. It typically involves designing and executing test scenarios that exercise the system under controlled conditions. To approximate real-world conditions, scenarios are consist of static elements (e.g., buildings and side-walks) and dynamic ones (e.g., surrounding vehicles or pedestrians), creating a challenging and realistic driving environment. Because setting up diverse, and often dangerous, scenarios in the physical world is expensive and risky, driving simulators capable of generating virtual test environments have become increasingly central to ADS testing [1–3].

To safely execute driving scenarios, an ADS is embedded in a driving simulator, such as CARLA [4] or MetaDrive [5], which constructs the virtual environment defined by the scenario description. These simulators are essential for efficiently executing safety-critical and edge-case scenarios that would be costly or infeasible to reproduce on physical proving grounds. An essential (yet often implicit) premise of simulation-based testing is that executions are *deterministic*: given the same scenario and ADS under test, repeated executions should yield identical results.

However, prior evidence by Chance et al. [6] shows that executions in CARLA exhibit notable deviations in vehicle trajectories across repeated runs. They found that the intrinsic nondeterminism stems from the simulator's reliance on the underlying game engine (Unreal Engine 4) for physics and rendering calculations. Such observations raise concerns that the reliability of simulation-based testing may be undermined by unintentional nondeterminism within simulator executions.

To address these concerns, we previously investigated flaky tests (i.e., nondeterministically passing or failing) within autonomous driving simulations. We analyzed their occurrence and severity finding that over 31% of test scenarios can exhibit flakiness caused by unintentional simulator nondeterminism. The study has been published at the Flaky Test Workshop 2025 (FTW 2025) [7]. However that investigation abstracted out underlying causes of nondeterministic behavior, assuming they solely come from the simulator engine.

In this work, we challenge previous assumptions by adopting a holistic perspective of nondeterministic behavior in scenario-based ADS testing. First, we perform a temporal analysis to examine how low-level sources of nondeterminism propagate through execution traces across reruns. This allows us to identify the root causes of trace-level variability. Second, we examine the higher-level flakiness of scenario evaluations induced by underlying execution nondeterminism, characterizing both its prevalence and severity. Together we aim to answer the following research questions:

RQ1 How deterministic are simulation-based test executions?
RQ2 What is the root cause of non-determinism in test executions?
RQ3 How flaky are the scenario evaluation results?
RQ4 Which safety requirements are mostly affected by flaky tests?

RQ1 examines how nondeterministic driving scenario execution traces are across multiple reruns. Understanding how much execution traces vary across reruns is essential, because low-level deviations can accumulate and ultimately alter high-level behavior (e.g., leading to a collision or a lack thereof). RQ2 is to uncover the root causes behind such nondeterminism identified in RQ1. Understanding where variability originates would enable to assess its impact and consider appropriate countermeasures. RQ3 focuses on the extent to which scenario evaluations become flaky due to nondeterministic simulations. We consider a test scenario *flaky* if its evaluation yields different outcomes across reruns without any changes to either the scenario itself or the ADS under test. If no flaky scenarios are observed, the simulator can be considered reliable in producing consistent evaluation results. RQ4 further analyzes which safety requirements are most affected by flaky tests. This question is crucial: even if many scenarios exhibit flakiness, its practical impact may be limited if only a narrow set of requirements is affected, allowing the remaining requirements to be evaluated reliably.

Our evaluation results show that CARLA exhibits a high level of execution variability already within the first ten seconds of scenario execution, indicating that fully deterministic evaluations are not achievable in this environment. We found that this low-level nondeterminism causes 43.6% of test scenarios to become flaky. One of the most affected requirements is *"no collisions with other vehicles"*, arguably one of the most critical safety criteria for any ADS. In contrast, MetaDrive demonstrates a high level of repeatability across reruns. Simulations of 120 seconds of driving show strong similarity across retries. Only 3 out of 1000 test scenarios exhibited flakiness, indicating that while not perfectly deterministic, MetaDrive evaluations achieve a high degree of reproducibility.

The apparent causality of lower-level nondeterminism to test flakiness highlights the need of a holistic view of nondeterminism in scenario-based ADS testing.

To summarize, the main contributions of this work are:

(1) An empirical investigation of determinism in simulation execution traces.
(2) An evaluation of how such nondeterminism affects flaky tests in CARLA and MetaDrive, two widely used driving simulators.
(3) Guidelines to mitigate potentially flaky tests in simulation-based ADS testing.

This paper is structured as follows. Section 2 provides the necessary background on scenario-based ADS testing in simulated environments. Section 3 defines intentional and unintentional nondeterminism and their relationship with flaky tests. Section 4 outlines the evaluation subjects as well as answers said research questions. Section 5 discusses potential general causes and implications of nondeterministic simulations on ADS testing and provides guidelines for addressing unintentional nondeterminism in future works on ADS testing. Section 6 concludes the paper and includes ideas for future work.

2 Background

Automated Driving Systems (ADS) have seen rapid development in recent years. To ensure their safety, they are extensively tested before future deployment. Field testing (e.g., testing on the proving grounds) is the closest to the real-world application but is incredibly costly and possibly dangerous. Test engineers naturally leverage simulators that promise safe, cheap and reproducible evaluation.

Figure 1 (presented by Zhong et al. [1]) illustrates a general ADS testing workflow in simulated environments.

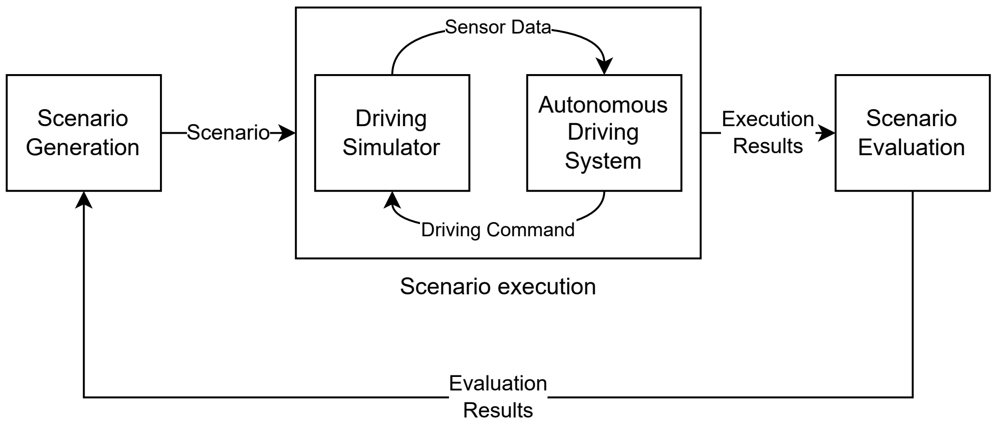


Fig. 1: An overview of scenario-based ADS testing in high-fidelity simulation. Adapted from Zhong et al. [1].

Simulation-based ADS testing typically starts with a *test scenario* (i.e., test input) that defines a sequence of events (e.g., trajectories of other vehicles from the start to the end of the scenario) happening in a specific environment (e.g., a T-junction with no traffic lights on a foggy day). In practice, a test scenario can be represented as an array of multiple variables, each representing different static and dynamic entities considered in the scenario.

Given a test scenario, a high-fidelity simulator, such as CARLA [4] or MetaDrive [5], creates a virtual world according to the scenario description. The ADS under test is embedded into the virtual world, fully controlling one of the virtual vehicles, commonly called as the *ego vehicle*. At each time step, the ADS under test takes as input *sensor data* generated by the simulator (e.g., images from virtual cameras mounted on ego vehicle) and outputs a *driving command* (e.g., throttle control and steering angle), which is then applied to the ego vehicle in the simulated world for the next time step. This closed-loop *scenario execution* (i.e., simulation) is discretized into small time steps (e.g., $0.05s$) and proceeds in a synchronous observation-action loop. The sequence of interleaved observations and actions over time step can be recorded to produce an execution trace $\tau = \{o_1, a_1, o_2, a_2, \ldots, o_n, a_n\}$, where o_i and a_i denote the observation and action at step i, respectively.

The simulation stops when the ego vehicle reaches its destination or when a predefined timeout occurs. At the end of the simulation, the scenario *execution results* (i.e., τ) are evaluated to see if the ego vehicle committed any safety violations (e.g., colliding with another vehicle or drifting out of its lane). Naturally, the expected test result is no violations (or fewer than a certain number of violations if there is a tolerance threshold).

The *evaluation results* (i.e., test outcomes) are often used to generate new test scenarios with the aim of maximizing objectives (e.g., degree of safety violations), therefore closing the testing loop.

Driving simulators often support deterministic simulations to achieve reproducible results [6]. Given the same input scenario and a deterministic driving system, scenario execution and evaluation results should be precisely the same in the deterministic simulation mode. However, not all simulations must be deterministic; some simulators might support intentional nondeterminism. To make things clear, we will further define and distinguish intentional and unintentional determinism in Section 3.

3 Simulation Nondeterminism and Flaky Tests

Simulation nondeterminism occur s when, given identical inputs, initial configurations, environmental conditions, and deterministic ADS, different behaviors are observed across simulation runs. In this section, we draw a distinction between *intentional* and *unintentional* nondeterminism in simulations, and discuss how they relate to flaky tests in ADS testing.

In some cases, simulators are *intentionally* nondeterministic. Random variation is introduced deliberately to capture realistic aspects of the modeled system. For example, in order to simulate loss of sensor data due to external perturbations, virtual LIDAR sensor may occasionally drop measurement points of low intensity. Similarly, background agents in driving simulators may follow probabilistic behavior models. Because human driver behavior cannot be fully predicted, such agents may probabilistically select actions from a distribution calibrated to real-world observations. However, simulations shall remain controllable for the sake of experimental reproducibility even when intentional nondeterminism is present. Random or probabilistic processes should therefore be governed by explicit random seeds to ensure that simulations can be replicated under identical conditions.

In other cases, simulators may be *unintentionally* nondeterministic. This can arise, for example, from simulator bugs or from external dependencies (e.g., the underlying game engine) that are themselves inherently nondeterministic. For instance, Chance et al. [6] note that concurrency and parallelization in game engines can lead to nondeterministic execution, as the order of operations cannot generally be guaranteed. Outside the domain of simulation-based testing, concurrency and parallelism in game engines are often a desirable trade-off, as they significantly improve rendering performance. However when it comes to testing ADS in simulations, such nondeterminism can lead to *flaky tests*, where repeated executions of the same test yield different outcomes (i.e., pass/fail results). From a testing perspective, intentional nondeterminism is easier to account for by fixing random seeds, allowing for degrees of tolerance in checks/assertions, or by making checks probabilistic. Unintentional nondeterminism, however, is difficult to control. In this paper, we focus on unintentional nondeterminism and its impact on flaky tests in ADS testing.

Note that flaky tests are based on scenario *evaluation* results in Fig. 1; even if scenario *execution* results vary due to unintentional nondeterminism in simulators, the corresponding scenario evaluation results could remain the same depending on safety requirements and tolerance thresholds used. For instance, if the ego vehicle takes slightly different paths in two test repetitions but runs a red light in each, both are evaluated as the same type of test failure. Therefore, in the following section we will independently examine lower-level nondeterminism in test *execution* results as well as higher-level flakiness in test *evaluation* results.

In software testing literature, various types of test flakiness have been defined based on their root causes. For example, Eck et al. [8] define infrastructure flakiness as test flakiness due to issues *outside* of project code (i.e., the ADS) but inside the execution environment (i.e., the driving simulator used to simulate scenarios using the ADS). Although it seems appropriate for describing the cause of flaky tests in our context, infrastructure flakiness is usually used to describe flakiness due to containers and/or the local host. Therefore, we use *simulator flakiness* to specifically refer to flaky tests caused by unintentional nondeterminism in driving simulators in this paper.

4 Empirical Evaluation

To recap, we aim to answer the following research questions on nondeterminism within test execution and test evaluations:

RQ1 How deterministic are simulation-based test executions?
RQ2 What is the root cause of non-determinism in test executions?
RQ3 How flaky are the scenario evaluation results?
RQ4 Which safety requirements are mostly affected by flaky tests?

The replication package for our case studies, as well as the analyzed data, can be found at https://figshare.com/s/b3c2bf419f76b095217a.

4.1 Subjects

To increase generalizability of our study, we conducted two independent case studies using two popular open-source autonomous driving simulators: CARLA [4] and MetaDrive [5]. Both are widely used in autonomous driving research, yet they differ substantially in terms of fidelity, realism, and the features they provide. We describe each case study in more detail below.

4.1.1 Simulator

We use the latest release of *CARLA 0.9.15* [4] with *Leaderboard 2.0* [9], which together constitute a state-of-the-art framework for benchmarking autonomous driving systems, widely used in contemporary research [10–16]. CARLA's scenario runner features a flexible interface that allows the embedding of driving systems for training and testing purposes. In CARLA, one can control various elements that make up a virtual environment, such as weather, lighting, traffic, road shapes, and surrounding buildings. To make the virtual environments more realistic, CARLA provides built-in, hand-crafted maps, ranging from complex city areas with roundabouts to long-stretching highways, including realistic static entities, such as traffic signs, buildings, and trees. This flexibility and controllability is needed for scenario-based testing, where testers try to identify sets of conditions that cause ADS under test to malfunction. By the default, CARLA sets the frame-rate (i.e., frequency of the world updates) to 20 frames-per-second (FPS).

We also use *MetaDrive* [5], an actively maintained open-source driving simulator capable of realistic perception [17]. MetaDrive is lightweight and allows to easily execute procedurally generated scenarios or built from an open-source dataset of vehicle trajectories. For an easier comparison we set the MetaDrive's world update frequency to match CARLA's i.e., 20 frames-per-second (FPS).

4.1.2 ADS under Test

For the ADS to run in CARLA, we use TransFuser++ v5 (TF++) [14], a state-of-the-art driving system, capable of driving in CARLA adapted to interact with Leaderboard 2.0 by Zimmerlin [16]. TF++ is an end-to-end driving model that takes input from sensors (i.e., a front RGB camera, Lidar and speedometer) and outputs steering control for throttle, braking, and lateral steering. TF++ takes advantage of transformer decoder [18] and path-based outputs, outperforming other models and setting new records on Longest6 [11] and LAV benchmarks [13] achieving a second place at the CVPR 2024 CARLA Autonomous Driving Challenge.

Unfortunately TF++, or any autonomous driving system originally developed for the CARLA simulator, cannot be executed in MetaDrive without substantial adaptation. The incompatibilities are due to the differences in underlying sensor models, data formats, and interface specifications between the two simulators. Since our objective is not to compare CARLA and MetaDrive, we simply use another driving system (agent) for MetaDrive. We use an *expert policy*, a simple, 3-layered neural network driving agent provided in MetaDrive by default. It inputs the ray-based LiDAR sensor data and outputs the throttle, braking, and lateral steering control. It is pre-trained using Proximal policy optimization (PPO) [19] reinforcement learning algorithm and is an integral component of the driving simulator.

4.1.3 Test Scenarios

To evaluate the frequency and severity of simulation flakiness, we require a benchmark set of test scenarios to execute in the simulator with the ADS under test.

For CARLA, we use the predefined scenarios provided by the *Bench2Drive* benchmark proposed by Jia et al. [20] to mitigate potential bias. Bench2Drive comprises 220 short routes (approximately 150 meters each) split across all twelve CARLA maps (commonly called *town*), each featuring a single safety-critical road event, e.g.,, loss of vehicle control on a slippery surface or a pedestrian emerging from between parked cars. Each *town* defines a unique static layout of the map, i.e., road network, traffic signs, lane markings, as well as surrounding buildings and parks to create a realistic environment. We have chosen this benchmark as it is the state-of-the-art validation suite among researchers developing driving systems with CARLA. Each scenario includes (1) a route the ADS under test must follow to reach its goal and (2) an event the ADS must face along a set route. For example, one scenario would specify that the ego vehicle must handle sudden lane changes of surrounding cars or pedestrians stepping out from parked cars. All scenarios include road users (e.g., cars, cyclists, and pedestrians) managed by a simulator built-in module, TrafficManager. At every simulation step, TrafficManager controls the behaviors of all road users. To prevent indefinite runs, the CARLA Leaderboard implements a timeout, i.e.,, a maximum duration allowed for the ego vehicle to reach its destination. If the vehicle fails to do so within this limit, the test is marked as failed, the simulation is terminated, and execution proceeds to the next scenario.

For MetaDrive, although we could not find a well-established benchmark like Bench2Drive for CARLA, we create a diverse set of test scenarios that would effectively evaluate the expert policy's performance across various driving conditions. We use a built-in procedural map generator (PG) to generate randomly seeded scenarios. PG generate a road network (a map) using unit blocks (e.g., straight road, curve, roundabout, intersection, T-junction) that can be further parametrized by length, the radius of curvature, etc. [5]. The simulator controls the traffic of background vehicles, with the traffic density set to its default value, where a few vehicles are initially spawned at each road block. By default, the initial position of the background vehicle is randomly seeded. Given a map, the ego vehicle is tasked with driving from the first roadblock to the last while keeping the lane and avoiding collisions with other vehicles. In total, we generated 1500 unique scenarios using different random seeds to ensure sufficient coverage of the procedural generation parameter space.

4.1.4 Safety Requirements

Safety requirements define the criteria for evaluating the behavior of the ADS under test during scenario execution. It varies between simulators due to differences in supported features and scenario complexity.

In CARLA, following the CARLA Leaderboard [9], we evaluate the ADS by counting the number of infractions corresponding to the following safety requirements:

RC1	no collisions with static objects;
RC2	no collisions with pedestrians;
RC3	no collisions with other vehicles;
RC4	no violations of red lights;
RC5	no violations of stop signs;
RC6	no out-of-lane driving;
RC7	no driving significantly slower than surrounding traffic;
RC8	no failure to yield to emergency vehicles;
RC9	no blocking of other vehicles;
RC10	no route deviations;
RC11	no route timeouts.

In MetaDrive, driving scenarios are simplified compared to CARLA's i.e., there are no pedestrians, cyclists and many surrounding static objects like stop signs. Therefore, we perform our evaluation by testing against the following safety requirements:

RM1	no collisions with other vehicles;
RM2	no collisions with sidewalk;
RM3	no out-of-lane episodes;
RM4	no route time-out.

4.2 RQ1: Nondeterminism in Scenario Executions

4.2.1 RQ1: Setup

To quantify the nondeterminism in scenario executions, we compute the similarity between multiple execution traces for the same scenario.

As introduced in Section 2, simulation-based testing yields an execution trace $\tau = \langle o_1, a_1, o_2, a_2, \ldots, o_n, a_n \rangle$, where each k-th element e_k ($k = 1, \ldots, 2n$) is either an observation o_i or an action a_i at simulation step i. For two traces of equal length, $\tau^1 = \langle o_1^1, a_1^1, o_2^1, a_2^1, \ldots, o_n^1, a_n^1 \rangle$ and $\tau^2 = \langle o_1^2, a_1^2, o_2^2, a_2^2, \ldots, o_n^2, a_n^2 \rangle$, we quantify their similarity through an element-wise comparison. Given that these elements are typically represented as high-dimensional vectors, we compute their similarity using normalised cosine similarity (via linear rescaling), ensuring each pairwise similarity is bounded within [0, 1] rather than [-1, 1]. The total similarity between two traces is then defined as the cumulative product of these individual similarities:

$$\mathcal{S}(\tau^1, \tau^2) = \prod_{k=1}^{2n} \frac{\mathit{CosineSim}(e_k^1, e_k^2) + 1}{2}$$

If τ^1 and τ^2 are identical, i.e., the execution is fully deterministic, all element-wise similarity values will be 1, and thus the cumulative similarity $\mathcal{S}(\tau^1, \tau^2)$ will also be 1. Otherwise, if there is any divergence between the traces, the cumulative similarity will be less than 1, indicating nondeterminism in the execution. Note that two traces may differ in length due to nondeterministic early termination of one execution (e.g., a crash or route deviation). In cases where two execution traces have different lengths, we truncate the longer trace to match the length of the shorter one. Specifically in case where $|\tau^1| = 2n$ and $|\tau^2| = 2m$, where $n < m$ we calculate cumulative similarity for the first n simulation steps.

Observation and action representations vary between simulators and driving agents, respectively. For CARLA, all available sensor data (LiDAR, RGB front camera, speedometer, GPS, and 6-axis inertial measurement unit) is used to represent observation vectors, while actions are represented as vectors of throttle, brake, and steering wheel angle. For MetaDrive, observations are represented as vectors of LiDAR sensor data, and actions are represented as vectors of throttle and steering wheel angle.

4.2.2 RQ1: Results

Figure 2 illustrates the cumulative similarity between two execution traces in CARLA and MetaDrive.

In both simulators, we observe a decline in cumulative similarity over time, indicating increasing nondeterminism as the simulation progresses. In other words, the nondeterministic factors in the simulation accumulate as the scenario unfolds. This trend seems intuitive, as small initial differences can amplify over time due to the complex dynamics of driving simulations. However, the rate and extent of this decline differ significantly between the two simulators.

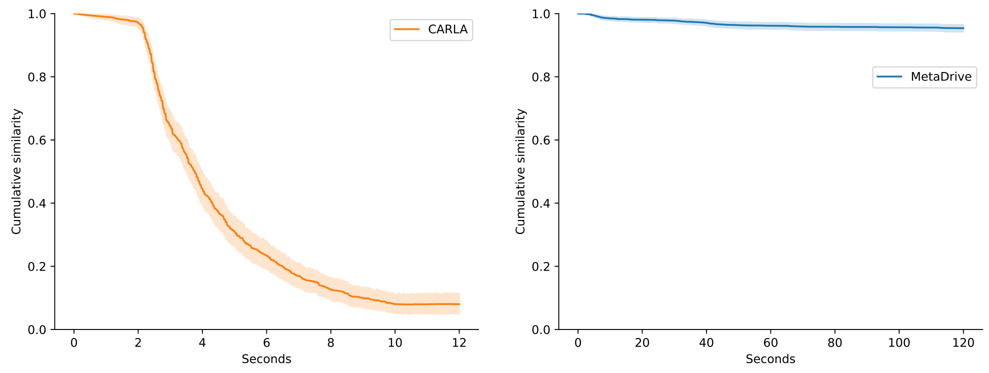


Fig. 2: Cumulative similarity of scenario execution traces over simulated time in CARLA and MetaDrive.

For CARLA, we observe a rapid decline in similarity as the scenario execution progresses, especially 2 seconds after the start of the simulation. The average cumulative similarity drops below 10% after just 12 simulated seconds, indicating a significant degree of nondeterminism in scenario executions. This implies that even short driving scenarios in CARLA cannot be reliably reproduced.

For MetaDrive, although there is also a decline in similarity over time, it occurs at a much slower rate compared to CARLA. After 120 simulated seconds, the average similarity is equal to 95.4%. This implies that MetaDrive allows for mostly reproducible scenario executions within a 2-minute timeframe.

It is worth recalling that the aim of RQ1 is not to compare the two simulators, but rather to assess the degree of nondeterminism in each. We cannot directly compare the two results, as the driving agents and scenarios used in each case study differ substantially. However, each case study is based on a representative configuration in the respective simulator, making the results relevant for researchers and practitioners using simulation-based testing.

The answer to RQ1 is that CARLA exhibits a high degree of nondeterminism in scenario executions, with similarity dropping below 10% after 12 seconds of simulation. In contrast, MetaDrive shows a much lower level of nondeterminism, maintaining a similarity of 95.4% after 120 seconds. In both cases, however, the level of nondeterminism increases over time.

4.3 RQ2: Identifying Nondeterminism Root Causes

4.3.1 RQ2: Setup

To identify the root causes of nondeterminism in scenario executions, we analyze the same observation–action traces $\tau = \langle o_1, a_1, o_2, a_2, \ldots, o_n, a_n \rangle$ previously collected to answer RQ1 (Section 4.2). For each scenario where we observed nondeterminism, we define the *root cause* of nondeterministic execution as the first element e_k ($k =$

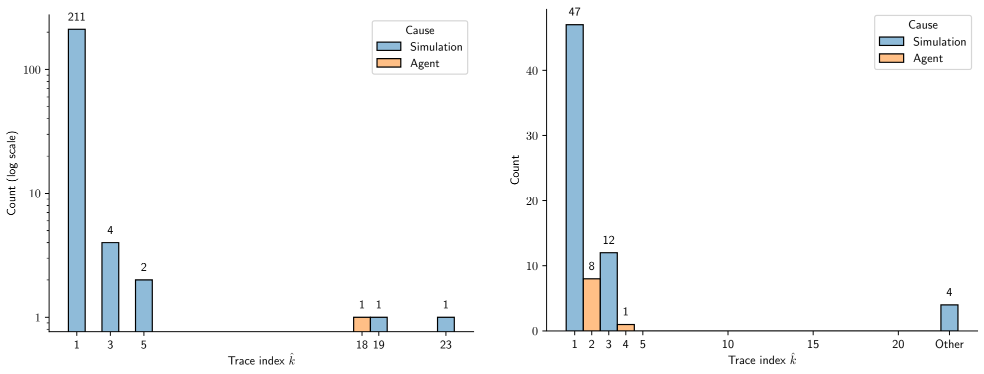


Fig. 3: Distribution of the first trace element at which nondeterminism occurs in scenario executions for CARLA and MetaDrive, illustrating the root causes of nondeterministic behavior.

$1, \ldots, 2n$) at which two traces begin to diverge (i.e., the first point where they are no longer identical). As in RQ1 (Section 4.2), we assess equality between trace elements using (normalised) cosine similarity.

More specifically, let $\hat{k}$ be the smallest index such that the similarity between the two elements $e^1_{\hat{k}}$ and $e^2_{\hat{k}}$ from traces τ^1 and τ^2 is less than 1. Then, we define $\hat{k}$ as the *root cause index* of nondeterminism for the given scenario. Depending on the value of $\hat{k}$, we can infer whether the source of nondeterminism lies in the driving simulator or in the driving agent, as follows:

- if $\hat{k} = 1$, then the first observation already differs between the traces, suggesting that the source of nondeterminism lies in the **simulation initialization**.
- if $\hat{k} = 2$, then the first dissimilarity occurs in the agent's actions, indicating that the **driving agent** exhibits nondeterministic behavior.
- if $\hat{k} > 2$, then the simulation execution is identical up to simulation step $\lfloor \hat{k}/2 \rfloor$, after which divergence begins.

More generally:

- When $\hat{k}$ is **even**, the dissimilarity occurs in the **observations**, implying that the nondeterminism originates from the **simulator**.
- When $\hat{k}$ is **odd**, the dissimilarity occurs in the **actions**, pointing to the **agent's** nondeterministic behavior as the root cause.

4.3.2 RQ2: Results

Figure 3 illustrates the root causes of nondeterministic scenarios identified in RQ1. Across both case studies, we observe that nondeterminism stems more frequently from the driving simulator than from the agent's actions. This result indicates that nondeterministic scenario executions (and the resulting test flakiness) are a systemic concern in simulation-based ADS testing, rather than an issue limited to specific ADS-simulator pairings.

For CARLA, all 220 scenarios exhibited some degree of nondeterminism. In 217 out of 220 cases, nondeterminism occurred within the first three observations, i.e., $\hat{k} \in \{1, 3, 5\}$, indicating that scenario initialization is the dominant cause of nondeterministic behavior. Notably, nondeterminism does not always manifest immediately after initialization. Three scenarios exhibited divergence later in the execution trace, specifically at $\hat{k} = 18, 19$, and 23. This observation suggests that, if the sources of nondeterminism during CARLA's initialization were mitigated, the simulator could support substantially more reproducible scenario executions.

In the MetaDrive case study, only 71 out of 1,500 executed scenarios exhibited some degree of nondeterminism. Among these, 58 scenarios became nondeterministic at the first or second observation generated by the simulator, i.e., $\hat{k} \in \{1, 3\}$, again indicating that scenario initialization is the most likely source of this unintended nondeterminism. In contrast, the driving agent was responsible for nondeterministic actions in only 9 scenarios, corresponding to trace indices $\hat{k} = 2$ and 4. Finally, four outlier scenarios exhibited nondeterminism at a later stage of the simulation due to differences in the observations received by the driving agent. These scenarios were manually inspected to identify potential causes related to scenario definitions; however, no consistent patterns were identified.

The answer to RQ2 is that, regardless of the simulator used, the most common cause of nondeterminism is scenario initialization in the driving simulator, occurring in 82% of MetaDrive scenarios and 99% of CARLA scenarios.

4.4 RQ3: Test Flakiness in Scenario Evaluations

4.4.1 RQ3: Setup

To quantify the frequency of flaky driving scenarios, we reevaluate each scenario ten times to assess the variability in its outcomes. As defined in Section 3, a test is considered *flaky* when identical scenarios yield different evaluation results across repeated simulation runs. To determine whether a scenario is flaky, we count the number of distinct "behaviors" observed during the repeated executions. If more than one unique behavior is observed, the scenario is classified as flaky.

We consider two test runs having the same "behavior" when the number of safety violations committed for each requirement is the same. For example, if a car runs a red light consistently in 10 repetitions, there is a single behavior observed (i.e., a non-flaky scenario). On the other hand, if we observe zero, one, and two red light violations in 3, 4, 3 repetitions, respectively, for a single scenario, we have three unique behaviors, and the scenario is marked as flaky.

Note that to assess test flakiness, we use a weaker notion of determinism (as opposed to comparing the simulations execution traces like in RQ1 and RQ2 in Sections 4.2 and 4.3), as it is directly connected with the safety requirements we test against. As noted in Section 3, scenario evaluation results, not execution results, matter the most. We do not require the simulator to be perfectly deterministic, as long as the infractions reported are consistent across repetitions.

To assess the flakiness of *CARLA*, we reuse the *Bench2Drive220* benchmark while embedding the *TransFuser++* driving agent. Each scenario is executed ten times, resulting in a total of 2200 simulation runs, each evaluated according to the defined safety requirements. Similarly, to evaluate the flakiness of *MetaDrive*, we procedurally generate one thousand unique driving scenarios and execute each ten times, yielding 10,000 test runs. Each execution is evaluated using the available safety requirements for the given simulator. Further details on the experimental setup are provided in Section 4.1.

4.4.2 RQ3: Results

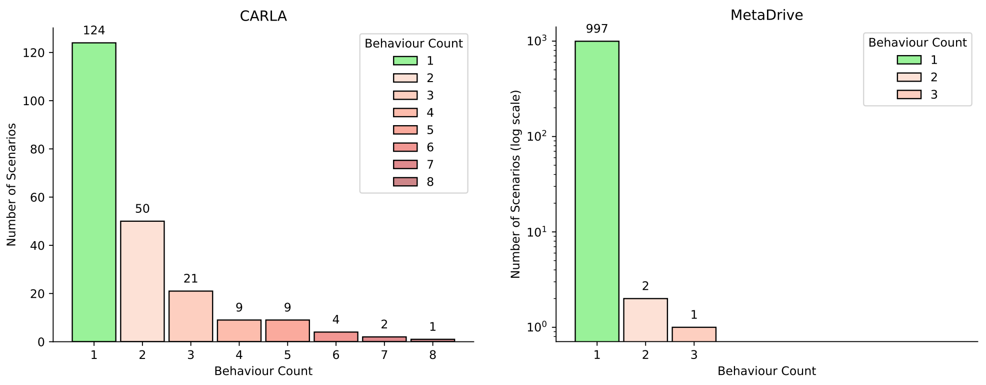


Fig. 4: Distributions of unique behaviors occurred when executing scenarios 10 times in CARLA and MetaDrive

Figure 4 presents the distribution of unique behaviors observed per unique scenario in CARLA and MetaDrive. When evaluating scenarios in CARLA, we find that only 124 out of 220 test scenarios exhibit a single unique behavior. This means that 43.6% (96/220) of the scenarios in this benchmark are flaky, which is quite surprising. Notably, the high nondeterminism in executions observed in Section 4.2 indeed correlates with nondeterminism in scenario evaluations. At the same time, a considerable portion (124/220) of scenarios remain non-flaky despite their corresponding nondeterministic underlying execution. Interestingly, 46 of the 96 flaky scenarios exhibit three or more unique behaviors, emphasizing the need for multiple reevaluations to obtain a complete picture of ADS behavior.

In contrast, the MetaDrive evaluations indicate that 997 out of 1,000 driving scenarios consistently produced a single unique behavior across ten repetitions. This suggests that flakiness is largely negligible when evaluating scenarios in MetaDrive. For completeness, we note that only 3 out of 1,000 scenarios exhibited two or more distinct behaviors, making such cases exceptionally rare.

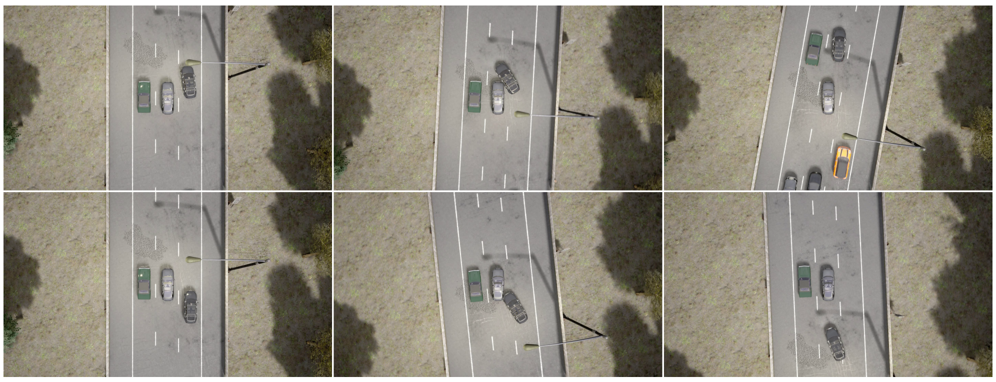


Fig. 5: Frames from two representative replays of CARLA Bench2Drive's "Scenario 17563", illustrating flaky behavior caused by nondeterministic scenario execution. The ego vehicle (a black sedan in the middle lane) encounters a cut-in scenario on a three-lane motorway. Columns correspond to time progression: the left, center, and right columns show the system state at times t_0, t_1, and t_2, respectively, where consecutive frames are separated by one second. Rows correspond to different replays. Due to nondeterministic behavior of the background vehicle to the right of the ego, the cut-in scenario unfolds differently across replays. In the first replay (**top row**), the adversary vehicle accelerates ahead of the ego and cuts into its lane, resulting in a severe collision that forces the ego vehicle to stop and fail the scenario. In the second replay (**bottom row**), the adversary vehicle remains behind the ego at the time of the cut-in, leading to a collision with lower relative velocity that allows the ego vehicle to recover and complete the scenario with no further violations.

The answer to RQ3 is that scenario flakiness substantially differs between simulators. In CARLA, 43.6% of the evaluated scenarios exhibit nondeterministic outcomes, with many showing multiple distinct behaviors across repeated runs. Scenario evaluations in MetaDrive demonstrate highly consistent results, with nearly all scenarios producing a single unique behavior across repetitions.

4.4.3 Manual investigation of scenario executions

To better understand how low-level nondeterminism within scenario execution relates to flakiness in evaluations, we manually investigated and recorded the five most flaky scenarios in CARLA and the three most flaky scenarios in MetaDrive. Video recordings of these replays are available in our replication package.

Figure 5 shows two replays of one of the most flaky CARLA scenarios, which exhibited six unique behavioral outcomes across ten repetitions.

For the given scenario, we observed that the variability comes from the interactions between the ego vehicle and an adversary background vehicle during a challenging cut-in scenario. Our manual investigation reveals that the scenario execution and its corresponding evaluation, diverges significantly due to the nondeterministic trajectory

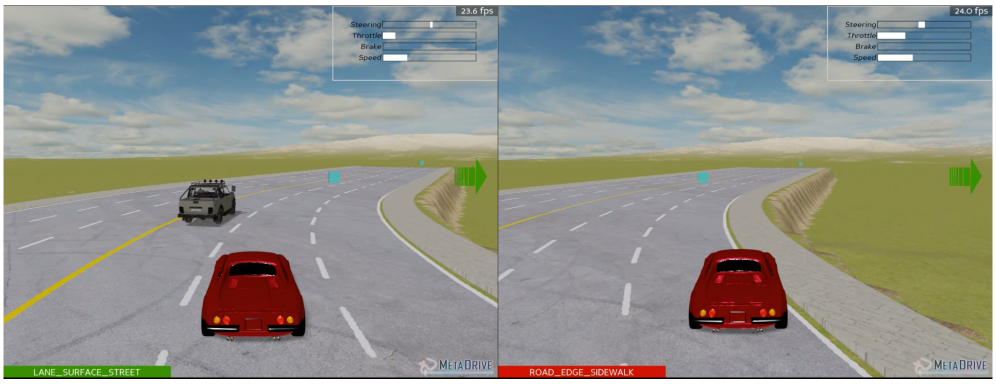


Fig. 6: Frames from two representative replays of MetaDrive's "Scenario 605", illustrating flaky behavior caused by nondeterministic scenario execution. The ego vehicle (a red sedan) takes two different paths in the final segment: in some repetitions it stays in the middle lane (**left image**), while in others it shifts to the right lane near the sidewalk (**right image**). When the vehicle moves close to the road edge, it collides with the sidewalk; when it remains in the middle lane, the scenario completes without any ego collision.

of the background vehicle, which results in collisions of varying dynamics. We suspect that these slight initial deviations in the background vehicle's trajectory trigger a butterfly effect, leading execution to have different outcomes.

Similarly, Figure 6 illustrates a flaky scenario in MetaDrive that exhibits two unique behaviors. In this case, the evaluation result is highly sensitive to the precise trajectory of the ego vehicle; depending on small variations in path planning, the execution can result in either multiple collisions with the sidewalk or a perfect run with no safety violations.

4.5 RQ4: Severity of Test Flakiness

4.5.1 RQ4: Setup

To answer RQ4, we investigate the degree of flakiness of each flaky scenario we found in RQ3 with regard to each safety requirement defined in Section 4.1.4. We quantify the degree by calculating the standard deviation for each infraction count across the ten repetitions. It implies how far each repetition result is from the average result. The higher the standard deviation, the more flaky the scenario is for the given safety requirement. Specifically, for each flaky scenario s and safety requirement r, we compute:

$$\sigma(s,r) = \sqrt{\frac{1}{n}\sum_{i}^{n}(\mathit{InfractionCount}_i(s,r) - \mu(s,r))^2}$$

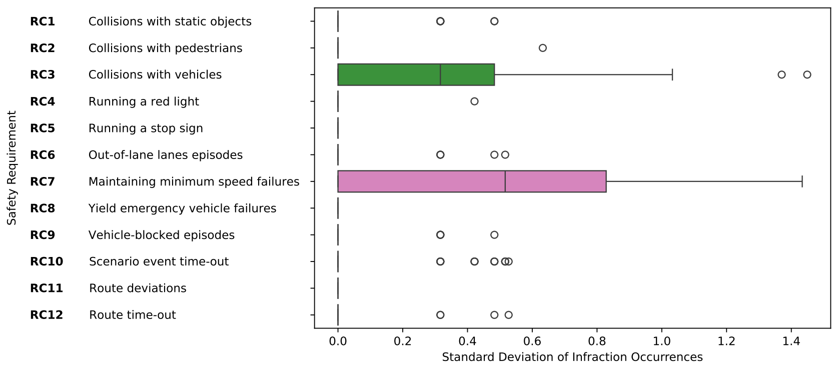
where $\sigma(s,r)$ is the standard deviation of s for r, $\mathit{InfractionCount}_i(s,r)$ is the infraction count of s for r at i-th repetition, $\mu(s,r) = \frac{1}{n}\sum_{i}^{n} \mathit{InfractionCount}_i(s,r)$

is the mean of the infraction counts of s for r, and $n = 10$ is the total number of repetitions.

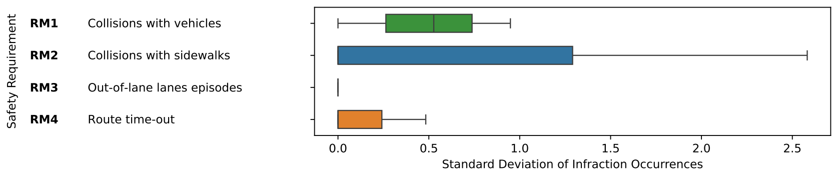
To give some intuition, consider a flaky scenario s_f and the *no collision with other vehicles* requirement r_v. For this scenario s_f we observed that for the first 5 executions there was no violation i.e., $InfractionCount_i(s_f, r_v) = 0$ for $i = 1, \ldots, 5$. However, for the last 5 execution the driving agent has collided with other vehicles twice i.e., $InfractionCount_j(s_f, r_v) = 2$ for $j = 6, \ldots, 10$. Then $\mu(s_f, r_v) = 1$, and therefore, $\sigma(s_f, r_v) = 1$. This means, on average, the difference between the infraction count for the flaky scenario of the given safety requirement is one.

CARLA and MetaDrive allow for the different testable Safety Requirements, see Section 4.1.4 for details.

4.5.2 RQ4: Results



(a) The degree of flakiness of 96 flaky test scenarios in CARLA.



(b) The degree of flakiness of 3 flaky test scenarios in MetaDrive.

Fig. 7: The degree of flakiness across flaky test scenarios found in CARLA and MetaDrive for each safety requirement violation, quantified as the standard deviation of infraction occurrences.

Figure 7a presents the standard deviation of the 96 flaky driving scenarios identified in RQ3, grouped by safety requirement violation. Table 1 summarizes the corresponding descriptive statistics (minimum, mean, and maximum) of these standard deviation values.

Table 1: Statistics of the standard deviation values for each infraction type in CARLA and MetaDrive

Safety Requirement	Infraction Type	Min	Mean	Max
RC1	Collisions with static objects	0.00	0.03	0.48
RC2	Collisions with pedestrians	0.00	0.01	0.63
RC3	Collisions with vehicles	0.00	**0.32**	1.45
RC4	Running a red light	0.00	0.00	0.42
RC5	Running a stop sign	0.00	0.00	0.00
RC6	Out-of-lane lanes episodes	0.00	0.02	0.52
RC7	Maintaining minimum speed failures	0.00	**0.47**	1.43
RC8	Yield emergency vehicle failures	0.00	0.00	0.00
RC9	Vehicle-blocked episodes	0.00	0.02	0.48
RC10	Scenario event time-out	0.00	0.06	0.53
RC11	Route deviations	0.00	0.00	0.00
RC12	Route time-out	0.00	0.02	0.53
RM1	Collisions with vehicles	0.00	0.49	0.95
RM2	Collisions with sidewalks	0.00	0.86	2.58
RM3	Out-of-lane lanes episodes	0.00	0.00	0.00
RM4	Route time-out	0.00	0.16	0.48

In CARLA, for most safety requirements (i.e., all except **RC3** and **RC7**), the mean degree of flakiness remains below 0.1. This indicates that, for the majority of safety requirements, the observed variability across repeated executions is minimal. Consequently, these requirements can be considered reliably testable in CARLA without substantial concern about flaky test outcomes.

However, **RC3** (*"no collisions with other vehicles"*) exhibits a mean standard deviation of 0.32. Given that **RC3** represents one of the most critical safety requirements, such a high degree of flakiness suggests that evaluating this criterion in CARLA may be unreliable due to nondeterministic interactions between vehicles.

Similarly, **RC7** (*"no driving significantly slower than surrounding traffic"*) shows the highest mean standard deviation of 0.47. This implies that across repeated scenario executions, the relative behavior between the ego vehicle and surrounding traffic varies considerably. Such variability likely reflects nondeterminism in CARLA's traffic simulation.

For completeness, we also analyzed the degree of flakiness for the three flaky scenarios identified in the MetaDrive study. As shown in Figure 7b, their overall variance is higher than in CARLA. Nevertheless, as discussed in RQ3, the overall frequency of flaky scenarios in MetaDrive is so low that this higher variance is not a practical concern.

The answer to RQ4 is that CARLA exhibits a high degree of flakiness for two safety requirements: **RC3** (i.e., no collisions with other vehicles, mean standard deviation 0.32) and **RC7** (i.e., maintaining minimum speed, 0.47), indicating nondeterminism that can affect the reliability of test evaluations for these safety requirements.

5 Discussion

In Section 4, we examined and confirmed the existence of nondeterministic scenario executions and resulting flaky test evaluations.

Although RQ2 examined root causes of nondeterministic execution traces for each driving scenario, we go one more step to discuss the potential causes of such nondeterministic simulations in general. We then discuss the implications of flaky tests in simulation-based ADS testing for triaging risks that have been under-appreciated. We also present strategies to mitigate such potentially flaky tests.

5.1 General Causes of Nondeterministic Simulations

Based on our observations, along with existing research on game engines used in driving simulators [6, 21], we propose potential causes of nondeterministic simulations: asynchronous behavior, floating point, time, and game engines.

5.1.1 Asynchronous Behavior

Comprehensive driving simulators are complex software systems that schedule and coordinate many tasks, such as 3D rendering, physics simulation, and background traffic control [17]. Many of those tasks can be executed simultaneously, e.g., reading sensor data while calculating vehicle trajectories, making them nondeterministic. To avoid such nondeterministic simulations, they often implement a stepped (synchronous) mode, where the simulation is discretized into smaller steps. Using the steps, all tasks can be scheduled in order, e.g., reading sensor data followed by calculating vehicle trajectories.

However, even in stepped mode, improper implementation can still lead to unintentional nondeterminism. For instance, CARLA 0.9.15 contains a known issue in its collision detection oracle, which may explain the high flakiness observed for the corresponding safety requirement in Section 4.5.2. Such hidden sources of nondeterminism emphasize the critical importance of testing and validating the simulators themselves, not only the autonomous systems under evaluation, to ensure reproducible and trustworthy experimental results.

5.1.2 Floating Point

Simulators rely on floating-point arithmetic, which can lead to subtle inconsistencies due to rounding errors and non-associative operations [22]. We suspect that such small numerical deviations, particularly in the positions or trajectories of surrounding vehicles controlled by the simulator (e.g., the `TrafficManager` in CARLA), may trigger a butterfly effect. This can result in a high degree of flakiness for safety requirements involving multiple vehicle interactions, such as **RC7** discussed in Section 4.5.2.

5.1.3 Time

Another common pitfall that can introduce nondeterminism is a dependency on 'system time' (i.e., real-world time) instead of 'in-simulation time' to schedule code execution. This can lead to nondeterminism due to hardware dependencies. For example, a low-spec machine might take 10 seconds of the real world to simulate 1 second of the virtual world, whereas another high-spec machine might take only 2 seconds of the real world for the same simulation. Therefore, using real time to schedule threads that coordinate the simulation processes can be a source of unexpected nondeterminism.

5.1.4 Game Engines

High-fidelity driving simulators often leverage game engines [17], such as Unity Engine used by LGSVL [23] and Unreal Engine 4 used by CARLA 0.9.15 [4]. These game engines were specifically designed and optimized for game and movie production applications. For example, they might prioritize 3D rendering performance over determinism. Chance et al. [6] empirically confirmed that some degree of nondeterminism is unavoidable in game engine-based simulators and extensively discussed the shortcomings of game engines in driving simulators.

5.2 Implications of Simulator Flakiness

To better understand the implications of flaky tests in simulation-based ADS testing, let us consider the critical scenario generation problem, one of the most widely studied problems in simulation-based testing [2, 3].

Critical scenario generation is to automatically generate scenarios that are likely to expose the critical safety violations of the ADS under test. A driving simulator runs the generated scenarios and evaluates the degree of safety violations they expose. If generated scenarios are flaky, the evaluation results may be unreliable. For example, a flaky test scenario may expose a critical safety violation in one run but not in the other. During the scenario generation process, this can decrease the effectiveness of scenario generation approaches, which often rely on the evaluation results of the previously generated scenarios to guide the generation of new scenarios. At the end of the scenario generation process, it can also lead to an under- or overestimation of the effectiveness of the scenario generation approaches, possibly leading to incorrect conclusions when comparing different approaches.

The same implications can be extended to other simulation-based testing problems, such as test scenario selection and prioritization [24], that rely on the evaluation results of potentially flaky scenarios due to nondeterministic simulations. Amini et al. [21] empirically examined this issue in the context of search-based testing using two simulators and ADS types. Their findings demonstrate that test flakiness can substantially distort the behavior of search-based scenario generation algorithms.

It is important to emphasize that these observations do not invalidate existing simulation-based studies or their results. The results may still be valid depending on the frequency of flaky scenarios and the degree of flakiness. However, researchers and practitioners should remain aware of the potential impact of nondeterminism and interpret evaluation outcomes with appropriate caution.

To address that, the following subsection presents a four-step mitigation strategy aimed at reducing the impact of nondeterministic evaluations and improving the reliability of simulation-based testing.

5.3 Possible Mitigation Strategy

To mitigate the risks of flaky tests in simulation-based ADS testing, we propose the following strategy: (1) *Acknowledge*, (2) *Prepare*, (3) *Check* and (4) *Respond*.

5.3.1 Acknowledge the flakiness

First of all, it is important to *acknowledge* that driving simulators can be flaky. For example, despite the flaky behaviors of CARLA shown in Section 4, most of existing CARLA-based studies do not report flaky test results, understandably due to their unawareness of simulator flakiness. Acknowledging the possibility of flaky tests allows us to, for example, allocate time before designing experiments to check and respond to any flaky tests, which will be discussed below.

5.3.2 Prepare for flakiness

Secondly, we should *prepare* for potentially flaky tests. This involves using the latest release of the simulator and running regression tests provided with the simulator to assess its determinism. For example, the latest CARLA release claims to support improved deterministic simulations, especially in traffic management, and includes a few smoke tests for determinism. Running these tests before the main experiments can help identify simulator flakiness. If these tests include some set-up steps, e.g., applying correct settings, ensuring the simulator is in deterministic mode, and seeding randomness, be sure to include them in your main experiment. However, these tests might not be inadequate to guarantee deterministic simulations for all possible scenarios. Thus, the next steps will be useful.

5.3.3 Check for flakiness

As a third step, we should *check* for flaky tests. This can be done by running each test scenario multiple times and assessing both the frequency and degree of its flakiness. Note that not all scenarios are necessarily flaky, and the frequency and degree of flakiness may vary among scenarios. Therefore, it is important to check as many scenarios as possible. For instance, one could randomly generate n scenarios and run each scenario r times to evaluate flakiness prior to the main experiments. While a larger n and r yield better results, simulation time and available resources should be considered. As a rule of thumb, we recommend that n and r be at least 30 and 10, respectively. This means the checks should ideally be completed within approximately 25 hours if each scenario takes 5 minutes to run. If potentially flaky scenarios are found, it may be necessary to increase n and/or r to gain a clearer understanding of flaky tests. For example, if 7 unique behaviors are observed among 10 runs of one scenario, one should consider increasing r to at least double the number of observed unique behaviors.

5.3.4 Respond to flakiness

Lastly, we should *respond* to any flaky tests identified in the previous step. Ideally, every test scenario used in the experiments should be run multiple times, and the results should be compared using statistical tests to ensure reliability. For example, when comparing two scenario generation approaches, each generated scenario could be run 10 times, in addition to comparing multiple runs of each approach [25], using tests such as the Mann-Whitney U tests. However, given limited simulation time and resources, this may not always be feasible.

6 Conclusions and Future Work

One of the key requirements for any autonomous driving verification, performed in simulation is that the simulation can be deterministic. Deterministic simulation guarantees that the tests executed are not flaky.

In this paper, we empirically investigated simulations nondeterminism as well as its impact on flakiness of driving scenarios. We independently evaluated two open-source driving simulators, CARLA and MetaDrive. First, our study shows that scenario executions in both CARLA and MetaDrive suffer from nondeterminism. These nondeterministic factors accumulate as a scenario unfolds, and we found that the most likely source of nondeterministic behavior is the process of scenario initialization.

Second, we found that scenario executions in CARLA exhibit a high level of nondeterminism, resulting in a substantial proportion of flaky scenarios, 43.6% (96/220). This flakiness in scenario evaluation is particularly present with regard to testing *"no collisions with other vehicles"* and *"no driving significantly slower than surrounding traffic"* safety requirements. On the contrary, we found that MetaDrive is capable of highly deterministic executions, resulting in the vastly deterministic as well as non-flaky evaluation.

Lastly, we included discussion section in which we consider further underlying causes of nondeterministic simulations, their implications and possible mitigation strategy, where we provide guidelines to mitigate and respond to potentially flaky scenarios.

As future work, we will extend our study to further simulation platforms. Another avenue for future work is developing a statistical testing framework for evaluating driving systems that takes potential flakiness into account.

Acknowledgements

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No. RS-2025-02218761, 50%) and by the Engineering and Physical Sciences Research Council (EPSRC) [EP/Y014219/1]. Phil McMinn is supported, in part, by the EPSRC grant "Test FLARE" [EP/X024539/1].

References

[1] Zhong, Z., Tang, Y., Zhou, Y., Neves, V.d.O., Liu, Y., Ray, B.: A survey on scenario-based testing for automated driving systems in high-fidelity simulation. arXiv preprint arXiv:2112.00964 (2021)

[2] Zhang, X., Tao, J., Tan, K., Törngren, M., Sánchez, J.M.G., Ramli, M.R., Tao, X., Gyllenhammar, M., Wotawa, F., Mohan, N., Nica, M., Felbinger, H.: Finding critical scenarios for automated driving systems: A systematic mapping study. IEEE Transactions on Software Engineering **49**(3), 991–1026 (2023) https://doi.org/10.1109/TSE.2022.3170122

[3] Ding, W., Xu, C., Arief, M., Lin, H., Li, B., Zhao, D.: A survey on safety-critical driving scenario generation—a methodological perspective. IEEE Transactions on Intelligent Transportation Systems (2023)

[4] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: CARLA: An open urban driving simulator. In: Proceedings of the 1st Annual Conference on Robot Learning, pp. 1–16 (2017)

[5] Li, Q., Peng, Z., Feng, L., Zhang, Q., Xue, Z., Zhou, B.: Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning. IEEE transactions on pattern analysis and machine intelligence **45**(3), 3461–3475 (2022)

[6] Chance, G., Ghobrial, A., McAreavey, K., Lemaignan, S., Pipe, T., Eder, K.: On determinism of game engines used for simulation-based autonomous vehicle verification. IEEE Transactions on Intelligent Transportation Systems **23**(11), 20538–20552 (2022)

[7] Osikowicz, O., McMinn, P., Shin, D.: Empirically evaluating flaky tests for autonomous driving systems in simulated environments. In: 2025 IEEE/ACM International Flaky Tests Workshop (FTW) Proceedings (2024). Institute of Electrical and Electronics Engineers (IEEE)

[8] Eck, M., Palomba, F., Castelluccio, M., Bacchelli, A.: Understanding flaky tests: The developer's perspective. In: Proceedings of the Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE), pp. 830–840 (2019)

[9] Team, C.: Get started with Leaderboard 2.0. Online; accessed 09 November 2025 (2024). https://leaderboard.carla.org/get_started_v2_0/

[10] Shao, H., Wang, L., Chen, R., Li, H., Liu, Y.: Safety-enhanced autonomous driving using interpretable sensor fusion transformer. In: Conference on Robot Learning, pp. 726–737 (2023). PMLR

[11] Chitta, K., Prakash, A., Jaeger, B., Yu, Z., Renz, K., Geiger, A.: Transfuser: Imitation with transformer-based sensor fusion for autonomous driving. IEEE Transactions on Pattern Analysis and Machine Intelligence **45**(11), 12878–12895 (2022)

[12] Shao, H., Wang, L., Chen, R., Waslander, S.L., Li, H., Liu, Y.: Reasonnet:

End-to-end driving with temporal and global reasoning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 13723–13733 (2023)

[13] Chen, D., Krähenbühl, P.: Learning from all vehicles. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 17222–17231 (2022)

[14] Jaeger, B., Chitta, K., Geiger, A.: Hidden biases of end-to-end driving models. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 8240–8249 (2023)

[15] Zimmerlin, J., Beißwenger, J., Jaeger, B., Geiger, A., Chitta, K.: Hidden biases of end-to-end driving datasets. arXiv.org **2412.09602** (2024)

[16] Zimmerlin, J.: Tackling carla leaderboard 2.0 with end-to-end imitation learning. Master's thesis, University of Tübingen (2024)

[17] Li, Y., Yuan, W., Zhang, S., Yan, W., Shen, Q., Wang, C., Yang, M.: Choose your simulator wisely: A review on open-source simulators for autonomous driving. IEEE Transactions on Intelligent Vehicles (2024)

[18] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)

[19] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)

[20] Jia, X., Yang, Z., Li, Q., Zhang, Z., Yan, J.: Bench2drive: Towards multi-ability benchmarking of closed-loop end-to-end autonomous driving. Advances in Neural Information Processing Systems **37**, 819–844 (2024)

[21] Amini, M.H., Naseri, S., Nejati, S.: Evaluating the impact of flaky simulators on testing autonomous driving systems. Empirical Software Engineering **29**(2), 47 (2024)

[22] Gruber, M., Roslan, M.F., Parry, O., Scharnböck, F., McMinn, P., Fraser, G.: Do automatic test generation tools generate flaky tests? In: Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, pp. 1–12 (2024)

[23] Rong, G., Shin, B.H., Tabatabaee, H., Lu, Q., Lemke, S., Možeiko, M., Boise, E., Uhm, G., Gerow, M., Mehta, S., *et al.*: LGSVL simulator: A high fidelity simulator for autonomous driving. In: IEEE ITSC, pp. 1–6 (2020). IEEE

[24] Sadri-Moshkenani, Z., Bradley, J., Rothermel, G.: Survey on test case generation, selection and prioritization for cyber-physical systems. Software Testing,

Verification and Reliability **32**(1), 1794 (2022)

[25] Arcuri, A., Briand, L.: A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. Software Testing, Verification and Reliability **24**(3), 219–250 (2014)