Empirically Evaluating Flaky Tests for Autonomous Driving Systems in Simulated Environments

Olek Osikowicz
University of Sheffield

Phil McMinn
University of Sheffield

Donghwan Shin
University of Sheffield

***Abstract*—In Autonomous Driving Systems (ADS) testing, a test scenario is a pre-defined, specific sequence of events, including static entities (e.g., road shapes and traffic signs) and dynamic entities (e.g., traffic lights and the trajectories of surrounding vehicles). By creating an environment according to a test scenario and running the ADS under test in that environment, we can verify whether the ADS causes any safety violations (e.g., collisions with other vehicles) or not. Due to the high cost and risks associated with setting up test scenarios in the real world, simulation-based testing, which relies on driving simulators that can create various virtual driving environments, has gained significant attention. Since simulated environments can be more deterministic than the real world, simulation-based testing can provide non-flaky tests, i.e., the same test outcome for the same test scenario (and the same ADS), in theory. However, do we really have no flaky tests in simulation-based ADS testing?**

This paper empirically investigates flaky tests in simulation-based ADS testing using two widely used, open-source driving simulators: CARLA and MetaDrive. Our results show that, surprisingly, 31.3% of benchmark test scenarios are potentially flaky due to nondeterministic simulations in CARLA, whereas MetaDrive does not yield any flaky tests. We further discuss potential causes of nondeterministic simulations, implications of flaky tests in ADS testing, and practical strategies for mitigating flakiness in future works.

I. Introduction

To ensure the safety and reliability of Autonomous Driving Systems (ADS), test scenarios play an essential role in ADS testing. A test scenario is a pre-defined, specific sequence of events, including static entities (e.g., road shapes, traffic lights, and background buildings) and dynamic entities (e.g., traffic lights and the trajectories of surrounding vehicles and pedestrians). It defines an environment where the ADS under test can operate, and we can verify whether the ADS causes any safety violations (e.g., collisions with other vehicles or pedestrians) by operating the ADS in the environment. Considering the high cost and risks associated with setting up various (and possibly extreme) test scenarios in the real world, driving simulators that can create various virtual driving environments have been increasingly used in ADS testing [1, 2, 3].

The benefits of simulation-based ADS testing are more than low cost and risk; using deterministic simulations can effectively suppress *flaky tests*. In other words, we can expect the same test outcome (pass or fail) for the same ADS[1] when the same test scenario is used. For example, open-source driving simulators, such as CARLA [4] (a high-fidelity driving simulator based on Unreal Engine 4) and MetaDrive [5] (a lightweight-simulator developed for training reinforcement learning based driving agents), support deterministic simulations [6, 7]. We can, therefore, expect reliable, non-flaky tests using CARLA and MetaDrive.

However, as shown by Chance et al. [8], CARLA exhibits intrinsic nondeterminism due to its reliance on the game engine (Unreal Engine 4) for physics and rendering calculations. Specifically, their evaluation results on six test scenarios based on a T-junction road show that the deviation in the trajectories of vehicles and pedestrians before and after collisions across multiple simulations is significant at the centimetre level. This raises questions about flaky tests.

In this paper, we empirically investigate flaky tests in simulation-based ADS testing using CARLA and MetaDrive, two open-source driving simulators widely used in relevant studies. Specifically, we aim to answer the following research questions:

RQ1 How many test scenarios are potentially flaky due to simulators' nondeterminism?

RQ2 How flaky are test results from the potentially flaky test scenarios?

RQ1 is to understand to what extent driving scenarios are flaky due to (unintentionally) nondeterministic simulations. We call a test scenario *potentially* flaky if the scenario *can* pass or fail without changes to the test scenario itself and the ADS under test. If there are no potentially flaky tests, it implies that the corresponding simulator can provide reliable test results.

RQ2 is to investigate further the degree to which individual test scenarios are potentially flaky due to nondeterministic simulations. This is an essential question since even if there were many flaky test scenarios for a driving simulator, their degrees of flakiness is insignificant, making the potentially flaky tests relatively harmless.

Our evaluation results show that, on the one hand, over 30% of the scenarios executed in CARLA exhibit potentially flaky behaviors. In particular, collisions with other vehicles and route timeout are the two most frequent safety violation types in flaky test scenarios. Given the importance of collision-related safety requirements, the frequency of potentially flaky tests in CARLA is quite surprising. On the other hand, all test scenarios in MetaDrive exhibit no flaky behaviors.

We further discuss the potential causes and implications of nondeterministic simulations in simulation-based ADS testing. We also discuss steps to mitigate flaky tests to make simulation-based ADS testing more reliable.

[1]Although ADS might have nondeterministic behaviors, it can be easily controlled by fixing random seeds.

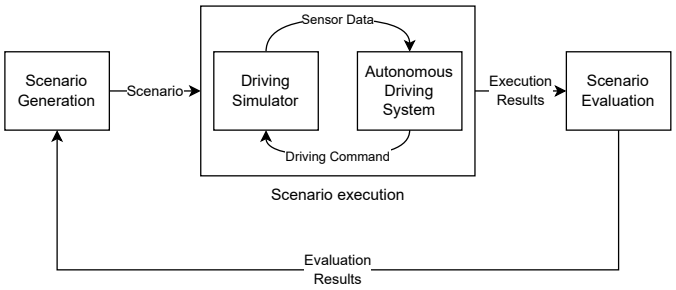

Fig. 1. An overview of scenario-based ADS testing in high-fidelity simulation. Adapted from Zhong et al. [1].

To summarize, the main contributions of this paper are:

(1) The empirical investigation of flaky tests in simulation-based testing with CARLA and MetaDrive, two widely used driving simulators in the research community.
(2) A discussion on the implications of potentially flaky tests on simulation-based ADS testing.
(3) Guidelines to mitigate potentially flaky tests in simulation-based ADS testing.

This paper is structured as follows. Section II provides the necessary background on scenario-based ADS testing in simulated environments. Section III defines intentional and unintentional nondeterminism and their relationship with flaky tests. Section IV contains two case studies with their methodology and results. Section V discusses potential causes and implications of nondeterministic simulations on ADS testing and provides guidelines for addressing unintentional nondeterminism in future works on ADS testing. Section VI concludes the paper and includes ideas for future work.

II. Background

A. ADS Testing in Simulated Environments

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. Test engineers naturally leverage simulators that promise safe, cheap and reproducible evaluation.

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

Simulation-based ADS testing typically starts with a *test (driving) 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 environment 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*. 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). To process sensor data and update the virtual environment, scenario execution is discretized into small time steps and runs into a synchronous state-action loop; at every step, the virtual environment is updated, and the ADS gets sensor data and outputs driving commands. The simulation stops when the ego vehicle reaches its destination or when a pre-defined timeout occurs.

At the end of the simulation, the results of the scenario run 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 more 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 [8]. 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 III.

III. Definitions

Nondeterminism in simulators is when, given the same inputs, starting configuration and environmental conditions, different behaviors are exhibited in different simulation runs. In this section, we draw a distinction between *intentional* and *unintentional* determinism.

In some cases, simulators are *intentionally* nondeterministic. This is because the reasons behind the simulated element (for example, an insect, an animal, or even a human) choosing to perform a particular behavior are not fully understood. In order to model these elements, therefore, a scientist or engineer may program the element to select randomly from a set of behaviors observed in real life, according to some probability distribution.

In other cases, simulators may be *unintentionally* nondeterministic. This could, for example, be due to bugs in the simulator or due to the use of components (e.g., the rendering engine) that themselves are nondeterministic for unaccounted and potentially undesirable reasons.

A software test that involves a nondeterministic simulator may be flaky for intentional or unintentional nondeterminism — test runs may involve different behaviors being exhibited, which, depending on the assertions or checks made, may lead to different test outcomes. 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, on the other hand, is hard to predict and allow for in tests. In this paper, we focus on flaky tests that result from unintentional nondeterminism in simulators. Such flaky tests are likely to be difficult to track down and debug.

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, we will focus on evaluation results rather than execution results in our evaluation in Section IV.

Luo et al. [9] performed the earliest empirical studies of test flakiness and categorize different causes. The reasons for unintentional nondeterminism in simulators may be due to several of the causes mentioned (e.g., due to the use of unpredictable thread orders and inaccuracies in floating-point computations) as discussed in V-A. However, from an ADS testing perspective, *infrastructure flakiness* is the most fitting type of flakiness for describing tests failing due to unintentional nondeterminism in driving simulators. Eck et al. [10] 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). Infrastructure flakiness is usually used, however, to describe flakiness due to containers and/or the local host. We argue in this paper, therefore, that *simulator flakiness* to be a new source of test flakiness due to unintentional nondeterminism in a simulator that is most apt for describing the type of flakiness encountered when testing ADSs evaluated in this paper.

IV. Evaluation

To recap, we aim to answer the two research questions on frequency of potentially flaky test scenarios and the degree of their flakiness.

We performed two independent case studies using two popular open-source autonomous driving simulation frameworks: CARLA [4] and MetaDrive [5].

The replication package for our case studies, as well as the analyzed data, can be found at https://figshare.com/s/36510668ad05ffa8c0bd.

A. Case study: CARLA

1) Simulator: To answer our research question, we use *CARLA 0.9.10.1* [4] with *Leaderboard 1.0* [11] which together form a popular framework to benchmark ADS used in many existing studies [12, 13, 14, 15, 16]. CARLA 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.

2) ADS under Test: For the ADS, we use TransFuser++ (TF++) [16], a state-of-the-art driving system, capable of driving in CARLA. 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 [17] and path-based outputs, outperforming other models and setting new records on Longest6 [13] and LAV benchmarks [15].

Upon further investigation, we concluded that TF++ does not contribute to flaky tests for the following reasons:

- Jaeger et al. [16] emphasized ensuring determinism during model training;
- The pre-trained model's weights during scenario execution are frozen, and the model is set in inference mode where we can control its randomness.

Therefore, we can safely conclude that flaky test results, if any, are due to unintentional nondeterminism in simulations.

3) Test Scenarios: To evaluate the frequency and the degree of simulation flakiness, we need a benchmark set of test scenarios to run in the simulator with the ADS under test. To mitigate any bias, we use pre-defined scenarios available in CARLA Leaderboard 1.0 [11], an official test benchmark which provides 25 challenging driving scenarios in three different maps called *Towns*. Each *town* defines the 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 most popular validation suite among researchers developing driving systems with CARLA. At the time of writing, the official leaderboard has received over 20 unique submissions.

Each scenario includes (1) a route the ADS under test must follow to reach its goal and (2) a list of events 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.

Since the original scenarios in the Leaderboard benchmark are too long (around 30 minutes to run each), we sliced them into smaller segments (around 5 minutes to run each), resulting in 128 scenarios. This is to reduce the execution time of each individual scenario while maintaining the challenging nature of the original. During the simulation of each (sliced) scenario, we set the timeout, as the CARLA Leaderboard does, to avoid an indefinite run if, for example, the ADS gets stuck. To answer our research questions, we repeated each scenario

Fig. 2. Sequence of frames from replaying "Scenario 8", the most flaky scenario found in CARLA. The ego vehicle passes through the street barrier in one of the scenarios due to a software bug.

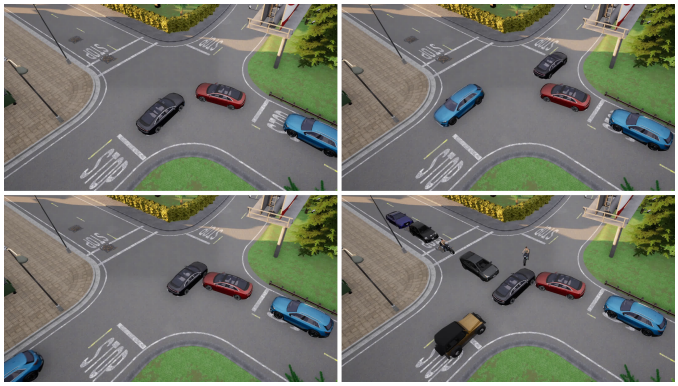

Fig. 3. Frames from replaying four representative repetitions of "Scenario 61" showing four unique behaviors. Each time, the ego vehicle (black sedan) approaches a four-way stop junction from one side and faces a red car that came on its way. Due to simulator nondeterminism, we can observe four alternative behaviors that happen next: **1) Top left**: the ego vehicle ends up in a deadlock where neither car can move, triggering a 'vehicle blocked' infraction. **2) Top right**: the ego vehicle manages to pass through the junction safely **3) Bottom left**: the ego vehicle tries to pass through but collides with the red car **4) Bottom right**: the ego vehicle ends up in a deadlock, causing more traffic, but tries to drive through, resulting in multiple collisions with both the red car and the motorcyclist.

simulation 10 times to see how scenario evaluation results vary from one rollout to another.

4) Safety Requirements: For each scenario, we evaluate the ADS by counting the number of infractions for the following safety requirements in CARLA: (RC1) no collisions with static objects; (RC2) no collisions with pedestrians; (RC3) no collisions with other vehicles; (RC4) no out-of-lane episodes; (RC5) no running a red light; (RC6) no running a stop sign; (RC7) no block-other-vehicles episodes; (RC8) no route time-out. We have chosen the above requirements as they capture the most important requirements of safe autonomous driving. The same requirements were used in CARLA Leaderboard [11], a popular driving benchmark.

5) Methodology: To answer RQ1, we count the number of potentially flaky scenarios among the 128 scenarios described in IV-A3. Recall that, as established in Section III, a flaky test happens when, given the same test scenario, different evaluation results are exhibited in different simulation runs. To check if a test scenario is potentially flaky, we count the number of unique "behaviors" observed during the repeated evaluation of the scenario. If the number of unique behaviors is greater than one, we consider that scenario potentially 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, we call it a single behavior (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 potentially flaky. We use this weak notion of determinism (instead of comparing the exact state of the simulation at all timestamps), as it is directly connected with the safety requirements we test against. As noted in Section III, scenario evaluation results, not execution results, matter. We do not require the simulator to be perfectly deterministic as long as the infractions reported are consistent across repetitions.

To answer RQ2, we investigate the degree of flakiness of each potentially flaky scenario we found in RQ1. We quantify the degree by calculating the standard deviation for each infraction count across 10 repetitions. It implies how far each repetition result is from the average result. The higher the standard deviation, the more potentially flaky the scenario is for the given safety requirement. Specifically, for each potentially flaky scenario s and safety requirement r, we compute: $\sigma(s,r) = \sqrt{\frac{1}{n}\sum_i^n (\mathit{InfractionCount}(s,r)_i - \mu(s,r))^2}$ where $\sigma(s,r)$ is the standard deviation of s for r, $\mathit{InfractionCount}(s,r)_i$ is the infraction count of s for r at i-th repetition, $\mu(s,r) = \frac{1}{n}\sum_i^n \mathit{InfractionCount}(s,r)_i$ is the mean of the infraction counts of s for r, and $n = 10$ is the total number of repetitions. For example, consider a potentially flaky scenario s_e and the no collision with pedestrians requirement r_p where $\mathit{InfractionCount}(s_e, r_p)_i = 0$ for $i = 1, \ldots, 5$ and $\mathit{InfractionCount}(s_e, r_p)_j = 2$ for $j = 6, \ldots, 10$. Then $\mu(s_e, r_p) = 1$, and therefore, $\sigma(s_e, r_p) = 1$. This means, on average, the difference between the infraction count of the potentially flaky scenario for the given safety requirement and the average infraction count is one.

6) Results: Figure 4 shows the distribution of unique behaviors observed. Our results show that only 88 out of 128 test scenarios exihibit a single unique behavior. It means that 31.3% (40/128) of the benchmark scenarios are potentially flaky in CARLA. It is worth noting that our notion of flaky tests is "weaker" than simulation determinism since we rely on scenario evaluation results instead of scenario execution results, as discussed in Section IV-A5. This implies that test evaluation results could have been flaky for around one-third of any scenarios in the CARLA Leaderboard benchmark [11], which is quite surprising. Interestingly, 12 of the 40 potential risk scenarios even have three or more unique behaviours. To better understand the nature of the potentially flaky scenarios, we manually investigated the 12 scenarios which exhibit three or more unique behaviors. For the most potentially flaky

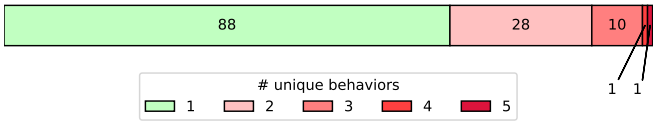

Fig. 4. Distribution of unique behaviors observed when evaluating TF++ [16] using CARLA [4] 10 times on 128 benchmark test scenarios.

scenario (i.e., "Scenario 8" exhibiting 5 different behaviors), we found that it is due to a bug[2] that allows the ADS to pass through a street barrier. Figure 2 presents scenario replay. We observed that the bug actually occurs at every repetition, but the evaluation results vary due to the collision detection sensor counts in CARLA. While the car's behavior looks consistent (the ego vehicle always passes through the barrier), CARLA reports a different number of collisions, between zero and five.

Figure 3 shows another potentially flaky scenario, "Scenario 61", exhibiting four different unique behaviors. For the given scenario, we found that the ego vehicle approaching challenging encounters at the junction can take different actions due to slight differences in vehicle positioning across multiple simulations. We suspect slight differences caused by simulator nondeterminism can trigger a butterfly effect, leading cars to exhibit completely different behaviors.

In the CARLA case study, the answer to RQ1 is that 31.3% (40/128) of scenarios are potentially flaky, showing a surprisingly high rate of potentially flaky test scenarios.

Figure 5 shows the standard deviation of the 40 potentially flaky scenarios from RQ1 for each infraction count. Table I summarizes the key statistics (min/mean/max) of the standard deviation values.

For most safety requirements (i.e., RC2, RC4, RC5, RC6, RC7), the mean degree of flakiness is less than 0.05. RC1 is shown to have relatively high mean degree of flakiness of 0.07, but this results is highly skewed due to a few outliers like "Scenario 8" described earlier. This implies that the degree of flakiness for the potentially flaky scenarios is indeed quite small for many safety requirements, making those safety requirements reliably testable in CARLA without worrying about flaky tests.

However, we have RC3 "no collisions with other vehicles" exhibiting a mean standard deviation of 0.3. RC3 is arguably one of the most essential safety requirements, yet the result implies it would be too unreliable to test in CARLA due to the high degree of flakiness. RC8 "no route time-out" also exhibits a mean standard deviation of 0.09, but it might be ignorable with a reasonable tolerance threshold for deciding a test failure.

[2]We reported the issue to CARLA leaderboard maintainers: https://github.com/carla-simulator/leaderboard/issues/185.

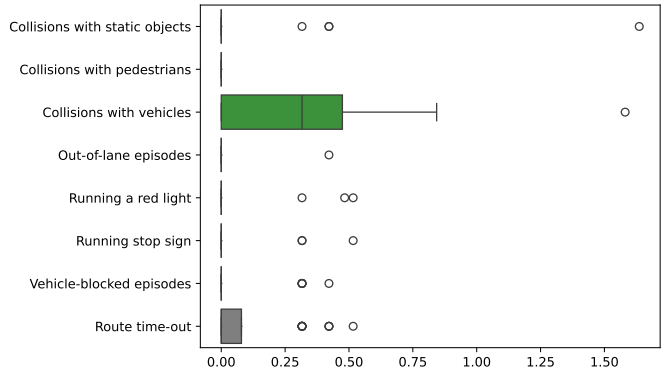

Fig. 5. The degree of flakiness of 40 potentially flaky test scenarios in CARLA for each infraction type.

In the CARLA case study, the answer to RQ2 is that CARLA results in a high degree of flakiness, i.e., a mean standard deviation of 0.3, for one of the most important safety requirements, i.e., no collisions with other vehicles. This calls for paying attention to test scenario evaluation results in CARLA.

B. Case study: MetaDrive

1) Simulator: For a second case study, we use *MetaDrive* [5], an actively maintained open-source driving simulator capable of realistic perception [18]. MetaDrive is lightweight and allows to easily generate scenarios or to build them from open-source trajectories datasets.

2) ADS under Test: TF++, or any other ADS developed for CARLA, is not easily executable with MetaDrive. 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 sensor data (Lidar, side scanner, and lane line detector) and outputs the throttle, braking, and lateral steering control. It is pre-trained using Proximal policy optimization (PPO) [19] reinforcement learning algorithm.

3) Test Scenarios: For the test scenarios, we use a built-in procedural map generator (PG) to generate 200 randomly seeded scenarios. PG generate a road network (a map) using unit blocks (e.g., straight road, curve, and a roundabout) that can be further parametrized by length, the radius of curvature, etc. [5]. The simulator controls the traffic of background vehicles, the density of which we set to the default value. 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.

4) Safety Requirements: In MetaDrive, driving scenarios are simplified compared to CARLA's i.e., there are no pedestrians, cyclists and many static objects. Therefore, we perform our evaluation by testing against the following safety requirements in MetaDrive: (RM1) no collisions with other

TABLE I
STATISTICS OF THE STANDARD DEVIATION VALUES FOR EACH INFRACTION TYPE IN CARLA

Infraction Type	Min	Mean	Max
(RC1) Collisions with static objects	0.00	0.07	1.64
(RC2) Collisions with pedestrians	0.00	0.00	0.00
(RC3) Collisions with vehicles	0.00	0.30	1.58
(RC4) Out-of-lane episodes	0.00	0.01	0.42
(RC5) Running a red light	0.00	0.03	0.52
(RC6) Running a stop sign	0.00	0.03	0.52
(RC7) Block-other-vehicles episodes	0.00	0.04	0.42
(RC8) Route time-out	0.00	0.09	0.52

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

5) Methodology: To answer RQ1 and RQ2, we use the same methodology as used in the CARLA case study (see Section IV-A5).

6) Results: Figure 6 shows the distribution of unique behaviors. All 200 test scenarios exhibit only one unique behavior when repeated 10 times, meaning that there are no potentially flaky tests.

To examine further how deterministic MetaDrive evaluation results are, we additionally investigated the test scenario execution results (i.e., the trajectories of all moving vehicles, including the ego vehicle, for all time steps). The result confirmed that MetaDrive is fully deterministic[3].

The answer to RQ1 and RQ2 in MetaDrive is that 100% (200/200) of test scenarios executed are non-flaky. Upon further investigation, we found that the simulation is fully deterministic regarding test scenario execution results.

V. DISCUSSION

In Section IV, we confirmed potentially flaky test scenarios in simulation-based ADS testing. Although we already know that they are due to nondeterministic simulations, we go one more step to understand the potential causes of such nondeterministic simulations below. 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.

A. Potential Causes of Nondeterministic Simulations

Based on our observations, along with existing research on game engines used in driving simulators [8], we propose potential causes of nondeterministic simulations: asynchronous behavior, floating point, time, and game engines.

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 [18]. Many of those tasks

[3]In fact, we found that the driving environment was not initializing properly when running multiple scenarios in sequence following the guideline. However, we could easily resolve it by explicitly re-initialising the environment before each scenario simulation. The issue is reported to the MetaDrive repository: https://github.com/metadriverse/metadrive/issues/758.

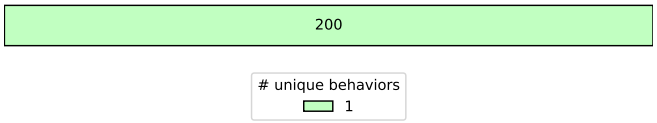

Fig. 6. Distribution of unique behaviors observed when evaluating MetaDrive's 'PPO expert agent' in MetaDrive [5] 10 times on 200 procedurally generated scenarios.

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 with the stepped mode, a potentially faulty implementation can still introduce unintentional nondeterminism. For example, CARLA 0.9.10.1 has a bug in the collision sensor that could generate multiple collision events per frame. We suspect this fault is responsible for the flaky behavior observed in executing "Scenario 8" described in Section IV-A6.

2) Floating Point: Simulators often involve floating point calculations, which can have unexpected results, such as non-associative addition [20]. We suspect these slight differences in the positions of surrounding vehicles controlled by simulators (e.g., TrafficManager in CARLA) can trigger a butterfly effect resulting in a crash observed in "Scenario 61" presented in Figure 3.

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.

4) Game Engines: High-fidelity driving simulators often leverage game engines [18], such as Unity Engine used by LGSVL [21] and Unreal Engine 4 used by CARLA [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. [8] 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.

B. 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 [22], that rely on the evaluation results of potentially flaky scenarios due to nondeterministic simulations.

It is worth noting that this implication does not nullify the value of existing simulation-based studies and their evaluation results. The results may still be valid depending on the frequency of flaky scenarios and the degree of flakiness. However, it is important to be aware of the issue of potentially flaky test scenarios and consider it when interpreting the results.

After conducting our research, we found a closely related study of Amini et al. [23], empirically evaluating the impact of flaky simulations on automated testing. They have also acknowledged that flakiness is a persistent issue in simulation-based ADS testing, highlighting the importance of mitigation strategies, which will be discussed in Section V-C.

C. 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*.

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 IV-A, 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.

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.

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.

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 [24], using tests such as the Mann-Whitney U tests. However, given limited simulation time and resources, this may not always be feasible.

VI. 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 frequency and degree of potentially flaky driving scenarios in two open-source driving simulators: CARLA and MetaDrive. We identified the potentially flaky driving scenarios by observing infractions that the driving system is committing in benchmark scenarios. We found that over 30% of driving scenarios executed in CARLA are potentially flaky due to unintentional nondeterminism. On the contrary, we found that MetaDrive is capable of fully deterministic execution resulting in non-flaky evaluation. Lastly, we included discussion section in which we consider potential causes of nondeterministic simulations, their implications and possible mitigation strategy, where we provide guidelines to mitigate 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 UK 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]. For the purpose of open access, the author has applied a Creative Commons Attribution (CC BY) license to any Author Accepted Manuscript version arising.

References

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

[2] X. Zhang, J. Tao, K. Tan, M. Törngren, J. M. G. Sánchez, M. R. Ramli, X. Tao, M. Gyllenhammar, F. Wotawa, N. Mohan, M. Nica, and H. Felbinger, "Finding critical scenarios for automated driving systems: A systematic mapping study," *IEEE Transactions on Software Engineering*, vol. 49, no. 3, pp. 991–1026, 2023.

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

[4] 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*, 2017, pp. 1–16.

[5] Q. Li, Z. Peng, L. Feng, Q. Zhang, Z. Xue, and B. Zhou, "Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning," *IEEE transactions on pattern analysis and machine intelligence*, vol. 45, no. 3, pp. 3461–3475, 2022.

[6] CARLA Contributors, "CARLA simulator documentation," https://carla.readthedocs.io/en/0.9.10/, 2024, online; accessed 27 October 2024.

[7] MetaDrive Contributors, "MetaDrive documentation — MetaDrive 0.1.1 documentation," 2024, online; accessed 27 October 2024. [Online]. Available: https://metadrive-simulator.readthedocs.io/en/latest/

[8] G. Chance, A. Ghobrial, K. McAreavey, S. Lemaignan, T. Pipe, and K. Eder, "On determinism of game engines used for simulation-based autonomous vehicle verification," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 11, pp. 20 538–20 552, 2022.

[9] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov, "An empirical analysis of flaky tests," in *Proceedings of the Symposium on the Foundations of Software Engineering (FSE)*, 2014, pp. 643–653.

[10] M. Eck, F. Palomba, M. Castelluccio, and A. Bacchelli, "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)*, 2019, pp. 830–840.

[11] CARLA Team, "Get started with leaderboard 1.0," 2020. [Online]. Available: http://leaderboard.carla.org/get_started_v1/

[12] H. Shao, L. Wang, R. Chen, H. Li, and Y. Liu, "Safety-enhanced autonomous driving using interpretable sensor fusion transformer," *preprint arXiv:2207.14024*, 2022.

[13] K. Chitta, A. Prakash, B. Jaeger, Z. Yu, K. Renz, and A. Geiger, "Transfuser: Imitation with transformer-based sensor fusion for autonomous driving," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 11, pp. 12 878–12 895, 2022.

[14] H. Shao, L. Wang, R. Chen, S. L. Waslander, H. Li, and Y. Liu, "Reasonnet: End-to-end driving with temporal and global reasoning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 13 723–13 733.

[15] D. Chen and P. Krähenbühl, "Learning from all vehicles," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 17 222–17 231.

[16] B. Jaeger, K. Chitta, and A. Geiger, "Hidden biases of end-to-end driving models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 8240–8249.

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

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

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

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

[21] 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 *IEEE ITSC*. IEEE, 2020, pp. 1–6.

[22] Z. Sadri-Moshkenani, J. Bradley, and G. Rothermel, "Survey on test case generation, selection and prioritization for cyber-physical systems," *Software Testing, Verification and Reliability*, vol. 32, no. 1, p. e1794, 2022.

[23] M. H. Amini, S. Naseri, and S. Nejati, "Evaluating the impact of flaky simulators on testing autonomous driving systems," *Empirical Software Engineering*, vol. 29, no. 2, p. 47, 2024.

[24] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Software Testing, Verification and Reliability*, vol. 24, no. 3, pp. 219–250, 2014.