

ANTI-CARLA: An Adversarial Testing Framework for Autonomous Vehicles in CARLA

Shreyas Ramakrishna^{1§}, Baiting Luo^{1§}, Christopher B. Kuhn², Gabor Karsai¹, and Abhishek Dubey¹

Abstract—Despite recent advances in autonomous driving systems, accidents such as the fatal Uber crash in 2018 show these systems are still susceptible to edge cases. Such systems must be thoroughly tested and validated before being deployed in the real world to avoid such events. Testing in open-world scenarios can be difficult, time-consuming, and expensive. These challenges can be addressed by using driving simulators such as CARLA instead. A key part of such tests is adversarial testing, in which the goal is to find scenarios that lead to failures of the given system. While several independent efforts in testing have been made, a well-established testing framework that enables adversarial testing has yet to be made available for CARLA. We therefore propose ANTI-CARLA, an automated testing framework in CARLA for simulating adversarial weather conditions (e.g., heavy rain) and sensor faults (e.g., camera occlusion) that fail the system. The operating conditions in which a given system should be tested are specified in a scenario description language. The framework offers an efficient search mechanism that searches for adversarial operating conditions that will fail the tested system. In this way, ANTI-CARLA extends the CARLA simulator with the capability of performing adversarial testing on any given driving pipeline. We use ANTI-CARLA to test the driving pipeline trained with Learning By Cheating (LBC) approach. The simulation results demonstrate that ANTI-CARLA can effectively and automatically find a range of failure cases despite LBC reaching an accuracy of 100% in the CARLA benchmark.

I. INTRODUCTION

Advancements in Machine Learning (ML) have led to considerable progress in the levels of autonomy achieved by Autonomous Vehicles (AVs) [1]. However, recent accidents such as Tesla’s autopilot crashes [2] and the fatal Uber self-driving car accident [3] demonstrate that current AV systems still can fail. To address this, testing and validating such systems before deploying them into real-world operations has received increasing attention. To determine failure cases of a system before they happen on the road, it is crucial to generate adversarial test cases that can fail the system. For example, in our previous work [4]–[6], we observed that an operational scene with adverse weather conditions (e.g., complex lighting, heavy precipitation, heavy fog) could be an adversarial test case for a vehicle operating based on visual perception. However, testing the system under such conditions in the real world can be expensive, slow, and often infeasible. Hence, recent research has been increasingly focusing on using synthetic data from simulators such as

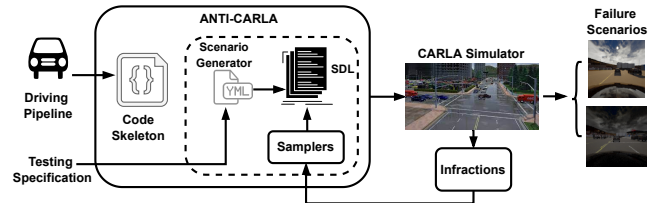


Fig. 1: Overview of ANTI-CARLA, the proposed adversarial testing framework integrated into the CARLA simulator. Given an arbitrary driving pipeline, the user specifies the desired test conditions, and the framework automatically generates adversarial test cases.

CARLA [7], AirSim [8], LGSVL [9], and Deepdrive [10]. CARLA is particularly focused on autonomous driving and is a widely used option in academia [4], [5], [11]–[13].

Traditionally, testing is restricted to a set of handmade scenarios with the goal of satisfying safety standards such as ISO26262 [14]. However, manually creating test cases is tedious and requires significant human labor. Recently, there has been substantial ongoing work in automating the test case generation process. Domain-Specific Modeling Languages (DSMLs) such as Scenic [15], and MSDL [16] are being developed for describing what a test case should contain. They provide a mechanism for describing the test conditions, the variable parameters, and their distribution, which is required for generating the test cases. These parameters are then sampled from their distribution ranges using state-of-the-art sampling algorithms such as random search, grid search, and Bayesian optimization search for generating suitable test cases. Despite this progress, the availability of frameworks that combine these components and allow straightforward test case generation is severely limited.

Only a few frameworks have been proposed that focus on generating test cases for autonomous driving [17]–[20]. These frameworks are built on custom simulators that are not open-source or available for use. Also, as shown in our previous work [5], the search mechanism (random and grid search) mostly used in these frameworks is inefficient in generating adversarial test cases. To address these issues, we propose ANTI-CARLA, a framework for automated adversarial testing, evaluation, and exploration of the performance of AVs within the open-source CARLA simulator. It provides a skeleton that allows for plugging in and testing any autonomous driving pipeline. It includes an Scenario Description Language (SDL) for describing the test conditions and a simple interface for specifying test conditions. The overall workflow of the framework is shown in Figure 1. Given a driving pipeline and testing specifications, ANTI-

¹Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN 37212, USA

²Department of Electrical and Computer Engineering, Technical University of Munich, 80333 Munich, Germany

[§]These authors contributed equally to this work.

CARLA automatically generates adversarial scenarios that fail the system. Due to its flexible and modular structure, any driving pipeline can be evaluated. The main contributions of this paper are as follows.

- We develop a domain-specific SDL that allows for modeling different testing scenarios for the AV, defined in terms of its operating conditions.
- We combine a mapping mechanism for an arbitrary AV system, test specification files, and an array of samplers into ANTI-CARLA to generate adversarial test cases automatically and efficiently.
- We use ANTI-CARLA to evaluate the state-of-the-art Learning By Cheating (LBC) controller [21]. Despite LBC achieving an accuracy of 100% in parts of the CARLA challenge, ANTI-CARLA generates several operating conditions that fail the controller.

The rest of this paper is organized as follows. In Section II, we summarize related research. We discuss the problem formulation in Section III and introduce the proposed framework in Section IV. In Section V, we evaluate the framework by generating fail cases for the LBC controller and analyzing them. Section VI concludes. The source code of this work will be published alongside this paper under <https://github.com/scope-lab-vu/ANTI-CARLA>.

II. RELATED WORK

In this section, we discuss related work. First, we introduce how test cases can be described and sampled. Then, we summarize existing frameworks that allow for generating test cases for a given driving pipeline. Finally, we summarize state-of-the-art driving pipelines for CARLA.

A. Test Case Description and Sampling

For testing software, test case generation has been a relevant research field for decades [22]. The field of AV testing has only recently started gathering interest. Domain-specific SDLs have been increasingly used for specifying the testing conditions [15], [23], [24]. For example, Scenic [15] is a popular language integrated with the CARLA simulator for setting up different scenarios. MSDL [16] is a language predominantly used in the industry to specify scenarios. Over the last years, languages such as SceML [24] have been developed to enhance the capabilities during testing.

To cover the operating conditions space, these languages are integrated with probabilistic samplers. Passive samplers such as the random and grid search are widely adopted to sample from the search space. These samplers do not use the feedback of previous results in the sampling process. For example, in grid search, a previously identified risky scenario is used as the initial condition, and the grid around it is searched to sample new scenarios [25]. Despite their wide usage, these samplers require prior information on the interesting regions of the search space (e.g., areas of high risk to the system under test) and can be labor-intensive. An array of active samplers has been proposed in recent years, which use the feedback of previous results to make the

sampling process more efficient. For example, sampling techniques such as incremental sampling, importance sampling, and adaptive sampling, borrowed from other fields such as uncertainty quantification and design space exploration [26]. Zhao et al. [27] used importance sampling to learn the parameters affecting the performance of the system under test, generating increasingly varied test cases that fail the system. The VERIFAI [23] software toolkit provides an array of passive samplers (e.g., random search, grid search, Halton sequence search [28]) and active samplers (e.g., Bayesian optimization search) to generate test cases.

We integrate both passive and active samplers into our framework. Due to the modular design of ANTI-CARLA, existing active samplers can be integrated easily.

B. Testing Frameworks

Simulation-based testing frameworks allow generating test cases for a given system. A framework consists of two main components: a mechanism for describing test cases, and a mechanism for generating test cases. Tuncali et al. [29] presented a testing framework built using their MATLAB tool called S-TaLiRo. The tool provides an optimization engine and a stochastic sampler to automatically sample test cases across the search space generated from the testing specification. The same authors later presented a simulation-based adversarial testing framework called Sim-ATAV [30], which performs adversarial testing in the scenario configuration space of perception-based AVs. Son et al. [19] proposed an Advanced Driver Assistance Systems (ADAS)-testing framework based on co-simulation of Siemens Amesim and Prescan software. The testing is driven by several handmade scenarios derived from safety standards such as ISO26262. The Paracosm framework [20] allows users to describe complex driving situations as programs. It provides a systematic approach for exploring the search space and contains a coverage metric to quantify the coverage. For Grand Theft Auto, a simulation-based harness for AV is available [17]. It includes a testbench that performs sampling using simulated annealing to sample the next simulation parameters based on the AV's performance in the current simulation. While all of these works focused on AV testing, they are implemented either in proprietary simulators or in limited custom scenarios.

To the best of our knowledge, no framework is readily available for testing AVs in open-source simulation. Despite CARLA being a widely used simulator across academia and industry, a flexible framework for testing AVs is currently unavailable for CARLA. This motivates the introduction of ANTI-CARLA for adversarial testing of arbitrary driving pipelines with minimal effort.

C. Autonomous Driving Pipelines

The proposed framework is designed to generate fail cases for a given autonomous driving pipeline. A range of controllers that perform well in CARLA is available. The Transfuser approach [31] uses transformers to fuse LIDAR and camera input. The World-on-Rails pipeline [32] only relies on camera images. It simplifies the driving

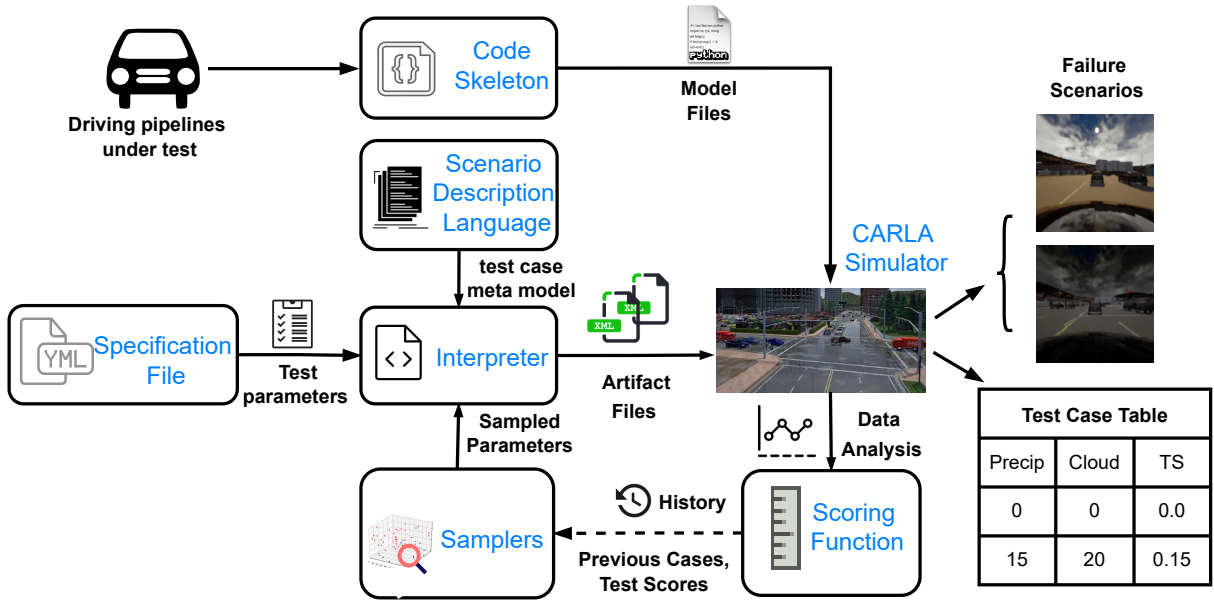


Fig. 2: Overview of the ANTI-CARLA framework for generating test cases that fail an AV system in the CARLA simulator. The driving pipeline and the test specifications are taken as inputs by the framework to generate failure test cases that can be post processed to analyze the problems with the system.

task by assuming that the vehicle does not influence the environment, then uses reinforcement learning to obtain a driving policy. Finally, the LBC approach [21] is also based on visual input. First, a teacher agent with complete access to internal simulator information is trained to imitate expert trajectories. Next, the actual controller is trained to imitate the teacher agent’s trajectories. By learning from a teacher who was “cheating” by having complete information, the student network is capable of learning highly accurate driving strategies. LBC achieved an accuracy of 100% on the original CARLA benchmark [21]. Current benchmarks are thus not always capable of identifying weaknesses of a controller, demonstrating the need for adversarial testing as offered by ANTI-CARLA.

While any controller could be used, we select LBC as a representative state-of-the-art approach to evaluate with ANTI-CARLA.

III. PROBLEM FORMULATION

Formally, the problem can be defined as follows. We have a simulator Sim that is given the current environment E_t , an AV with a driving controller C , and the current state of the AV $state_t$. The state of the vehicle $state_t \in \mathbb{R}^n$ describes the position, speed, and steering angle of the vehicle. The current environment E_t can be defined in terms of a set of temporal variables such as weather, traffic density, the time of day that change over time and a set of spatial variables that describe roadway features. These features are characterized by waypoints w denoted by a two-dimensional matrix of latitude and longitude. These waypoints can be mapped to different road segments that constitute a track on which the AV is tested. Further, the controller C is a function that perceives the environment using measurements from multi-modal sensors such as a camera, LIDAR, Radar,

etc. to compute actuation controls of speed, throttle, and brake. The control signals are sent to the simulator, which generates the next state of the vehicle $state_{t+1}$ by running the controller C under the given environment variables E_t : $Sim(state_t, E_t, C) = state_{t+1}$. An ordered sequence of n consecutive states $state_i, i \in \{t-n, \dots, t\}$ is considered a *scene*. The simulator has some infraction function I that records for each state its infractions $I(state_t) \in \mathbb{R}_n$. A scoring function TS assigns a score to the entire scene: $TS(scene) = \sum_{k=t-n}^t I(state_k)$. Then, we want to solve the following problem.

Problem. Given a simulator Sim , a driving controller C and some conditions on the environment E , generate a set of scenes that results in high infraction scores TS .

Addressing this problem requires a framework that allows for (1) specifying the operating conditions in which the system vehicle is expected to operate. (2) finding the conditions that could result in a high infraction score, and (3) integrating a given controller C into the given simulator Sim . In the next section, we introduce ANTI-CARLA to implement these requirements.

IV. ANTI-CARLA FRAMEWORK

In this section, we discuss the proposed ANTI-CARLA framework and its components shown in Fig. 2.

A. Scenario Generator

The first component of the framework is a scenario generator. It includes a Scenario Description Language for modeling scenarios as well as specification files for specifying and selecting testing parameters. We define a test case in terms of a scene, which is a time-series trajectory of the system’s path in the environment that lasts 30 s to 60 s. A

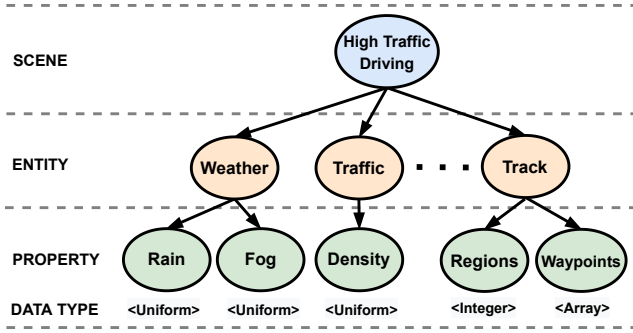


Fig. 3: Example for the structure of the proposed meta-model.

scene is represented using the environmental conditions (E) and the AV system’s parameters (A). E can be defined using structural features (e.g., type of road and road curvature) and temporal features (e.g., weather and traffic density). A includes information like the starting position, onboard sensors, and actuators. Together, E and A form the testing parameter set. The value for some of these parameters can be sampled from specified distributions. Further, the sampling process is governed by a set of physical constraints that limit the rate at which these parameters can evolve. For example, the time of day has a fixed rate at which it can change. Including these constraints during sampling results in more meaningful scenes, as shown in our previous work [5].

Scenario Description Language: We have designed an SDL in the textX [33] meta-language for modeling a scene. A grammar contains the rules for describing scene in the meta-language. A meta-model contains the actual description of a scene. A visualization of the structure of the meta-model is shown in Fig. 3. A scene s is a collection of entities $\{e_1, e_2, \dots, e_k\}$ that represent different environmental conditions and agent parameters. For example, the scene *High Traffic Driving* is specified by its *Track* as well as its *Weather* and *Traffic* conditions. Entities are further specified by a set of properties $\{p_1, p_2, \dots, p_m\}$. For example, the *Weather* can have the properties of *Rain* and *Fog*. Each property has a name n and a data type t . For example, *Rain* is a uniform distribution, while the *Waypoints* are an array of integers. Special data types such as distributions can again have properties, e.g. the range of a uniform distribution. The meta-model allows for a structured description of any desired scene in the CARLA simulator.

Specification Files: We also provide a set of specification files with the scene parameters that can be selected by the user. These files serve as an abstract representation of the SDL modeling concepts. Based on the user’s selection of the sampler, the selected parameters are sampled from their respective distributions. The remaining parameters are assigned default values. We divide the information into three specification files. First, the user selects a scene specifier. We show an excerpt of the scene specifier file in Figure 4. It specifies which town to use as a map, which track to drive, the distributions for the weather traffic density, pedestrian density, and the sampling constraints. The user

```
Scenario Description{
  town: 5 //Available towns 3 and 5
  track: 1 // 1 track available for each town
  regions: 5 //Each town has 5 regions
  weather: //Weather parameters and distribution range
  cloudiness: [0,100]
  precipitation: [0,100]
  time-of-day: [-90,90]
  pedestrian_density: [0,3]
  traffic_density: [0,10]
  Constraints: //A constraint on the rate of change in
    parameter values
    weather_delta: 2
    traffic_delta: 2
    pedestrian_delta: 1
  Infraction_Metrics: //Infraction metrics to be
    recorded
    Infraction Penalty: true
    Off-road Driving: true
    Route Deviation: false
  Record Frequency: 5Hz } //Frequency of data recording
```

Fig. 4: Excerpt of a scene specification file. The specification files describe the available inputs, the user then only has to select a value for each concept.

also specifies which infraction metric to use and at which frequency data should be recorded. Second, an agent specifier is needed, which includes agent-related information such as the available controllers, list of sensors and their positions, and the sensor data that can be recorded. Third, a sampler specifier determines which sampler to use from a list of available samplers.

Finally, the language has an interpreter as shown in Fig. 2. It connects the specification files, the SDL, and the probabilistic samplers discussed in the following section. First, the interpreter extracts the parameters that require sampling and the fixed parameters from the specification files. It then sends the parameters that need sampling to the sampler. The sampler generates a value for these parameters from their distribution ranges. Finally, the sampled parameters and the fixed parameters are parsed to the SDL meta-model to generate artifact files that drive the simulator.

B. Adapter Glue Code

The framework also requires a driving pipeline under test. Integrating different pipelines is not straightforward since they might not have the right interface to be used “as in” the framework. For example, driving pipelines developed outside of CARLA may not be directly used in the simulator because of strict interface requirements. They may need to be “adapted” to meet the interface expectations of the simulator [34]. To address this, we generate an adapter that interfaces the driving pipeline code with the sensors and actuators in the format required by the simulator’s API as shown in Figure 5. The adapter is synthesized from the agent specification file, which has a list of sensors required by the driving pipeline, its positions, and the sampling rates.

The adapter reads the available sensors from the autonomous agent class in the simulator’s API and extends it with the sensors requested in the specification file. Thus, a code structure for the requested sensors is generated and provided to the driving pipeline code. Also, the actuators required by the simulator are read from the API and made available in the code. With the required sensors and actuators

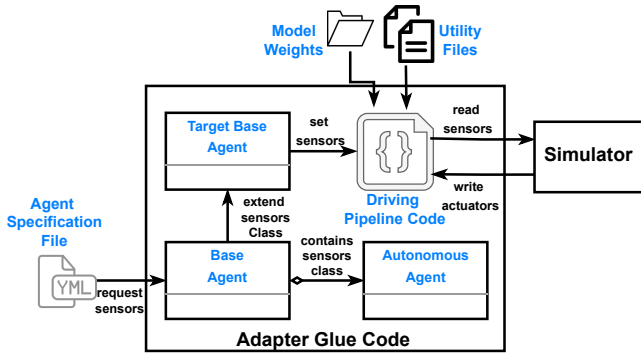


Fig. 5: An illustration of the adapter glue code that interfaces the sensors and actuators of CARLA to the code of the AV system.

available through the adapter, the user needs to provide the driving pipeline code. There are specific directories for any utility files and model weights of the controller, which are linked with the code skeleton. The user only needs to handle this interface for setting up their controller correctly. If the actuators and sensors are not properly set up, the simulator will throw an error. In future work, we will include an automatic check for ensuring a correct-by-construction setup between the code, sensors, and actuators.

C. Scoring Function

For evaluating the driving proficiency of the AV system in the generated scene, the framework also provides a scoring function called test score TS . The scoring is performed based on all of the infractions performed by the system in a given scene. Infractions measured in CARLA include route deviation, lane violation, traffic rule violation, running a stop sign, running a red light, and off-road driving. Each infraction I_k is assigned a weight w_k . In the current setup, these weights are set to the values used in the CARLA challenge [35]. However, they can be varied depending on the current use case. The infractions are then combined into the weighted score as shown in Equation 1:

$$TS = \sum_{k=1}^n w_k \cdot I_k \quad (1)$$

The test score TS generated from the infractions is stored along with the test case parameters in a test case table as shown in Figure 2. These test cases and the scores are used online to drive the active samplers towards regions of the search space that have previously resulted in high test scores. The table can also be used to perform an offline post-analysis to identify the failure conditions of a given controller, allowing to retrain and improve the controller.

D. Samplers

We have integrated several samplers to perform search-based test case generation. A sampler is interfaced to the SDL through an interpreter, which provides it with the scene parameters and the distribution from which different values for the parameters can be sampled.

We included two kinds of samplers available in the framework. First, we implemented several passive samplers

since they are fast and widely used. They do not use the feedback of previous results in the sampling process. *Random Search*, uniformly samples the parameter value from their respective distributions at random. *Grid Search* exhaustively searches all of the combinations of the parameters in a given grid. *Halton Sequence Search* [28] is a pseudo-random technique that samples the parameters using co-primes as their bases. While these samplers perform well, their non-feedback sampling approach results in a directionless search that could miss several important failure test cases. Further, they do not balance the exploration vs. exploitation of the search space, which is required for generating diverse failure cases [5].

To overcome these limitations, we have also included two adversarial samplers, *Random Neighborhood Search (RNS)* and *Guided Bayesian Optimization (GBO)* that we developed in our previous work [5]. These samplers use the feedback of the system’s previous performances when sampling the parameters for the current test case. In addition, they also capture constraints and correlations between the different test parameters to generate meaningful test cases. The overall idea with the feedback and the constraints is to move the sampling process towards the regions of the search space that are highly likely to fail the system.

The *RNS* sampler extends the conventional random search with the kd-tree nearest neighborhood search algorithm. This extension provides the random sampler with the capability to exploit. If the test case generated from randomly sampled parameters results in a high test score, the region around these parameter values is exploited. Otherwise, the parameters are again randomly sampled from the entire distribution. The *GBO* extends the conventional Bayesian Optimization sampler with constraints, which restrict the region where the acquisition function looks for the next sampling variables.

V. EVALUATION

In this section, we present several experiments to evaluate the usability of ANTI-CARLA for adversarial testing. We use the framework to compare the adversarial test cases generated by different samplers. Then, we compare the performance of the LBC controller to the Transfuser approach [31] on those test cases. The experiments were run on a desktop computer with AMD Ryzen Threadripper 16-Core Processor, 4 NVIDIA Titan XP GPUs, and 128 GiB RAM.

A. Simulation Setup

The proposed framework is integrated into the CARLA simulator. We used the CARLA challenge API [35] to create one closed loop track in both towns 3 and 5. We use LBC as the controller C under test. The geometry of the track is defined by ten waypoints as shown in Figure 6. We divide the track into several regions containing two waypoints each. For each track, we thus obtain five regions. We divided each track into regions to create shorter scenes in which we can vary the weather conditions as well as traffic and pedestrian densities. Each track can then have ten different environmental conditions. The length of the

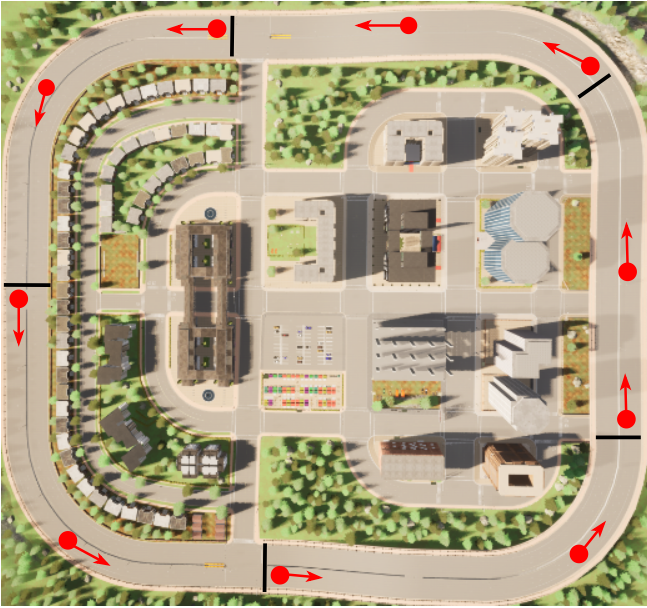


Fig. 6: An illustration of the track in CARLA’s town5. Red arrows highlight the waypoints and the five regions are separated by black lines.

track and number of regions can be specified in the scene description file and can thus be changed with minimal effort in the code. We use the API to create and control the traffic and pedestrians in each scene. For evaluating each simulation, we record the driving score, the route completion score, and the test score TS based on the list of infractions available from the API.

1) *Comparison Metrics:* Besides evaluating the performance of the driving pipeline, the framework also allows us to evaluate the performance of the samplers used to generate adversarial test cases. We use the following two metrics.

Failed Test Cases (FT): This score measures the efficacy of the samplers in generating failure test cases. It is calculated as the number of failed test cases N_{Fail} compared to the total number of sampled test cases N_{Total} :

$$FT(\%) = \frac{N_{Fail}}{N_{Total}} \cdot 100 \quad (2)$$

Total Execution Time: The overall time taken by the sampler to sample N_{Total} test cases and execute them in the simulator is a metric relevant for practice.

B. Results

We generated 100 test cases each for the track in town3 and town5. Each test case represents a scene that lasts between 30 seconds to 60 seconds. We varied the environment parameters of cloudiness c , precipitation p , time of day d , and traffic density t to generate different test cases. We varied the cloudiness and precipitation in the range $[0, 100]$, the time of day in the range $[0, 90]$, and the traffic in the range $[0, 50]$. We used the initial conditions of $d = 0^\circ$ (dusk), $c = 0^\circ$, $p = 0^\circ$, and $t = 5$. To score the test cases, we computed a test score as $TS = R_i \cdot I_S$. Here, R_i is the route completion percentage of the i^{th} route, and I_S is the weighted sum of major infractions such as collision with pedestrians, collisions with vehicles,

Samplers	Town	Failed Test Cases (%)	Test cases with collisions (%)	Test cases with infractions (%)
Random	3	17	9	10
	5	13	7	8
RNS	3	27	13	15
	5	21	10	13

TABLE I: Test case statistics for the random and RNS sampler.

collisions with static objects, and minor infractions such as running a stop sign, running a red-light signal, and off-road driving. The weights for these infractions are taken directly from the CARLA challenge setup. To drive the test case generation process, we selected the commonly used random sampler as a baseline and compared it against the RNS sampler. The simulator was run in synchronous mode at a fixed rate of 20 frames per second. If a test case includes either a collision, infraction, or route in-completion, it is considered to be a failure test case.

1) *Visualization:* First, we visualize several test cases by showing the frontal camera view from each scene. In Fig. 7, we show four exemplary test cases for LBC. Fig. 7-a is a nominal test case from town5 with low precipitation and dusk time of the day and low traffic. Fig. 7-b is a failed test case from town5 with high precipitation and traffic. A collision occurred when the controller tried to steer to the right lane. Fig. 7-c is a failed test case from town3 with no precipitation and low traffic. Here, the AV can be seen navigating a tunnel. Towards the end of the tunnel, the AV collides with a pillar. Finally, Fig. 7-d is another failed test case from town3, where the AV runs a red traffic light. These examples demonstrate that a diverse set of fail cases could be obtained.

2) *Sampler Comparison:* Table I shows the statistics of the collisions and infractions generated by the two samplers across all test cases. The RNS sampler generates a higher number of failed test cases than the random sampler. In general, the AV performed better in town5 than in town3. The track in town5 was shorter, had fewer traffic lights and stop signs, and did not have complex landmarks such as a tunnel or a round-about. The controller failed 13% and 21% of the test cases generated by the random and RNS sampler, respectively. Town3 has several traffic lights and a tunnel. Here, the controller failed 17% and 27% of the test cases generated from the random and RNS sampler, respectively. These fail cases occurred in the region that included the tunnel as well as areas with traffic signs and stop signs nearby. The framework required 140 minutes to generate the test cases for town5 and 325 minutes for town3 when using the random sampler. With the RNS sampler, the execution times are 176 minutes for town5 and 384 minutes for town3. This shows that the more efficient RNS sampler does not add significant overhead.

Fig. 9 shows the test cases sampled by the random and RNS sampler plotted in the operating conditions space. The failed test cases are marked in red, and the test cases that

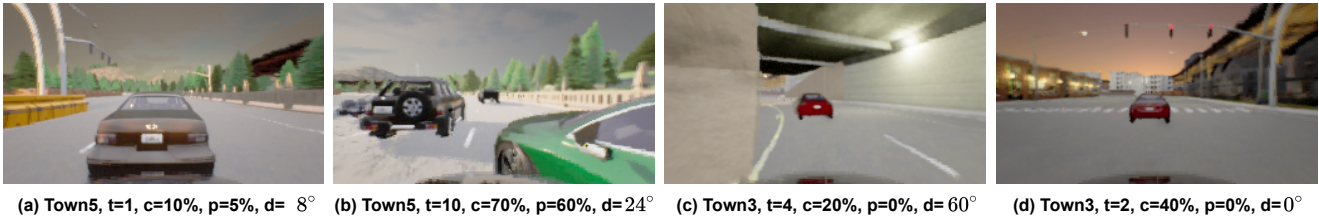


Fig. 7: Screenshots of the test cases as captured by the forward-looking camera of the AV. Descriptions of these scenes are provided below the images.

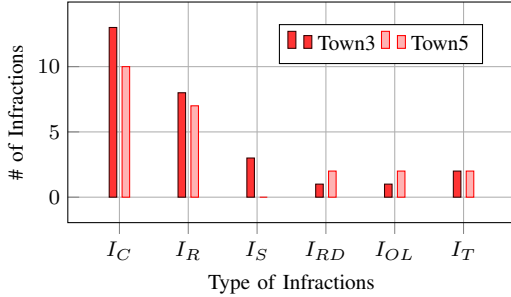


Fig. 8: Town-wise infractions of the LBC controller. The infractions included in the plot are: I_C includes collisions with vehicles, pedestrians, and static obstacles. Other infractions are running a red light I_R , running a stop sign I_S , route deviations I_{RD} , off lane driving I_{OL} , and simulation timeout I_T .

passed without failures or collisions are marked in blue. The random sampler randomly samples the test cases across the search space. This makes it hard to analyze common causes of the controller’s failures. In contrast, the failed test cases generated by the RNS sampler occur in clusters, which makes it easier to hypothesize the causes of the failures. The figure shows that the controller had failures mostly in high precipitation and dusk (time of the day) operating conditions.

3) *Town-wise Infractions*: Fig. 8 shows the infraction statistics of the LBC controller gathered from the 100 test cases generated by the RNS sampler across towns 3 and 5. I_C includes collisions with vehicles, pedestrians, and static obstacles. Other infractions are running a red light I_R , running a stop sign I_S , route deviations I_{RD} , off lane driving I_{OL} , and simulation timeout I_T . A timeout occurs if the vehicle is involved in an unrecoverable collision or if the controller operating the vehicle unexpectedly stops with a hard brake signal. The controller had a total of 27 failure test cases in town3. Among these, the vehicle was involved in a collision in 13 test cases, red-light violations in 7, stop-sign violation in 5, a route deviation in 1, off-lane driving in 1, and a timeout in 2 test cases. In comparison, the vehicle had fewer failure test cases in town 5. The vehicle had fewer collisions, red-light violations, and stop sign violations. However, there were more test cases in which the vehicle had route deviations and off-lane driving.

4) *Controller Comparison*: ANTI-CARLA can also be used to compare different controllers. We compare the performance of the Transfuser [31] controller to LBC using the track in town5. We ran the Transfuser and LBC for the same 100 test cases generated with the random sampler. The LBC and Transfuser controllers had 13 and 23 failed test cases,

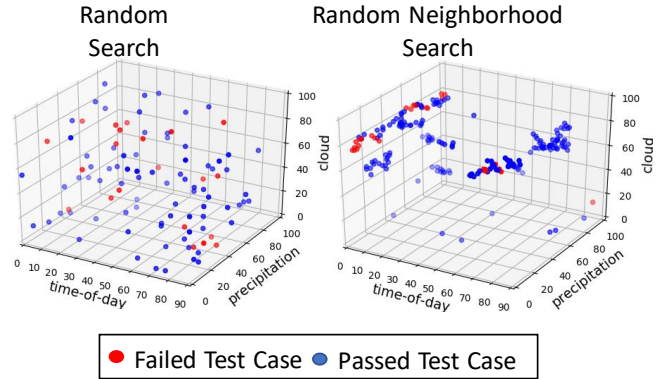


Fig. 9: Comparison of the 100 scenes sampled by the random and RNS samplers. The RNS sampler generates distinct clusters of failures. Plot axis: x-axis represents the time of day, y-axis represents the precipitation level and z-axis represents the cloud level.

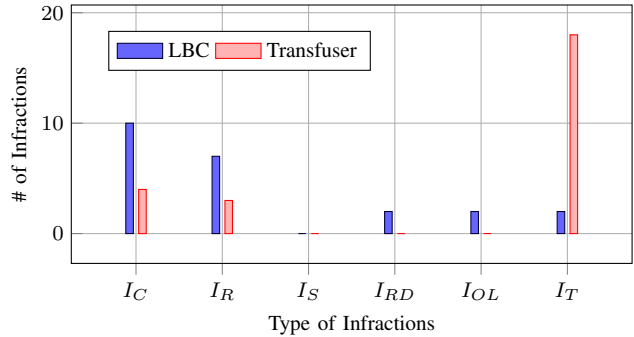


Fig. 10: Infraction breakdown caused by the LBC and Transfuser controllers. The infractions included in the plot are: I_C includes collisions with vehicles, pedestrians, and static obstacles. Other infractions are running a red light I_R , running a stop sign I_S , route deviations I_{RD} , off lane driving I_{OL} , and simulation timeout I_T .

respectively. Fig. 10 shows a breakdown of the infractions caused by these controllers. Simulation timeout is the main reason for the Transfuser’s higher failure rate. This could be due to the expert policy used during training not being sufficient for handling all scenarios. Besides the frequent timeouts, the Transfuser had significantly fewer collisions and infractions than LBC. Both controllers struggled with detecting red traffic lights. The authors of Transfuser suggested that this is due to traffic lights being placed on the opposite side of intersections, which is difficult to detect in camera images [31]. These results show how ANTI-CARLA allows us to identify the weaknesses of different controllers by focusing specifically on adversarial test scenarios.

5) *Recommendations*: The previous sections show that LBC tends to crash into the leading vehicle in heavy rain or in unusual lighting conditions, caused either by the time of day or by entering a tunnel. The highest infraction scores are obtained for adverse weather conditions and challenging landscapes such as traffic lights, tunnels, and roundabouts. By identifying those main reasons for failures, several suggestions can be made for improving LBC. First, adding another sensor modality could make the perception more robust to rain or darkness. For example, LIDARs are less susceptible to scene lighting levels or rain intensity. Second, the robustness of the camera-based perception could be improved by training the controller with more images taken in the adverse conditions of rain and at dusk.

VI. CONCLUSION

In this paper, we introduced the ANTI-CARLA framework for the CARLA simulator that allows to automatically generate test cases that fail an arbitrary AV system. Testing frameworks available in the literature are either tailor-made for a specific use case or built on proprietary simulators. In contrast, ANTI-CARLA is an open-source extension to an open-source simulator. The framework integrates a Scenario Description Language for modeling test scenarios in terms of the system's operating conditions, sensor, and actuator faults. A test specification file is used to specify and select the test conditions, infraction metrics, and samplers required for generating the test cases. The SDL is driven by an adversarial sampler that searches across the specified operating conditions space. We used this framework to test the popular Learning By Cheating controller and to compare different samplers. We also used ANTI-CARLA to compare the performance of LBC to the Transfuser approach, allowing us to identify the weaknesses of each controller.

We plan to move this research in several directions. First, currently, only static scenes can be sampled. A temporal sequence of scenes leading up to each fail case is unavailable. Temporal and dynamic sampling process using Reinforcement Learning and Monte Carlo tree search will be added in the future. Second, the sampling process is currently slow, with each test case taking approximately 5 minutes. To scale the testing process, we will parallelize the simulations and the sampling process across dockers. Fourth, the adversarial test cases could be used for a closed-loop controller training workflow as proposed in [36]. By training the controller with the obtained failure scenes and then sampling new failure scenes, the controller can be iteratively optimized.

Acknowledgment: This work was supported by the DARPA Assured Autonomy project and Air Force Research Laboratory. Any opinions, findings, and conclusions expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA or AFRL.

REFERENCES

- [1] S. Gibbs, "Google sibling waymo launches fully autonomous ride-hailing service," *The Guardian*, vol. 7, 2017.
- [2] B. Vlasic and N. E. Boudette, "'self-driving tesla was involved in fatal crash,' us says," *New York Times*, vol. 302016, 2016.
- [3] P. Kohli and A. Chadha, "Enabling pedestrian safety using computer vision techniques: A case study of the 2018 uber inc. self-driving car crash," in *Future of Information and Communication Conference*. Springer, 2019, pp. 261–279.
- [4] C. Hartsell, S. Ramakrishna, A. Dubey *et al.*, "Resonate: A runtime risk assessment framework for autonomous systems," in *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, may 2021.
- [5] S. Ramakrishna, B. Luo, Y. Barve *et al.*, "Risk-aware scene sampling for dynamic assurance of autonomous systems," *arXiv preprint arXiv:2202.13510*, 2022.
- [6] L. Habermayr, M. Hofbauer, J.-V. Zacchi *et al.*, "Situation-aware model refinement for semantic image segmentation," in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2021, pp. 2696–2702.
- [7] A. Dosovitskiy, G. Ros, F. Codevilla *et al.*, "Carla: An open urban driving simulator," *arXiv:1711.03938*, 2017.
- [8] S. Shah, D. Dey, C. Lovett *et al.*, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics*. Springer, 2018, pp. 621–635.
- [9] G. Rong, B. H. Shin, H. Tabatabaee *et al.*, "Lgsvl simulator: A high fidelity simulator for autonomous driving," in *23rd International conference on intelligent transportation systems*. IEEE, 2020, pp. 1–6.
- [10] Voyage, "Deepdrive simulator," 2020. [Online]. Available: <https://deepdrive.io/>
- [11] M. Hofbauer, C. B. Kuhn, G. Petrovic *et al.*, "Telecarla: An open source extension of the carla simulator for teleoperated driving research using off-the-shelf components," in *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2020, pp. 335–340.
- [12] C. B. Kuhn, M. Hofbauer, Z. Xu *et al.*, "Pixel-wise failure prediction for semantic video segmentation," in *2021 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2021.
- [13] M. Hofbauer, C. B. Kuhn, J. Meng *et al.*, "Multi-view region of interest prediction for autonomous driving using semi-supervised labeling," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–6.
- [14] P. Kafka, "The automotive standard iso 26262, the innovative driver for enhanced safety assessment & technology for motor cars," *Procedia Engineering*, vol. 45, pp. 2–10, 2012.
- [15] D. J. Fremont, T. Dreossi, S. Ghosh *et al.*, "Scenic: A language for scenario specification and scene generation," in *Proceedings of the 40th annual ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI)*, June 2019.
- [16] O. foretellix, "Open m-sdl," [Online]. Available: <https://www.foretellix.com/open-language/>
- [17] H. Abbas, M. O'Kelly, A. Rodionova *et al.*, "Safe at any speed: A simulation-based test harness for autonomous vehicles," in *International Workshop on Design, Modeling, and Evaluation of Cyber Physical Systems*. Springer, 2017, pp. 94–106.
- [18] C. E. Tuncali, G. Fainekos, H. Ito *et al.*, "Sim-ataw: Simulation-based adversarial testing framework for autonomous vehicles," in *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control*, 2018, pp. 283–284.
- [19] T. D. Son, A. Bhawe, and H. Van der Auweraer, "Simulation-based testing framework for autonomous driving development," in *IEEE International Conference on Mechatronics (ICM)*, vol. 1. IEEE, 2019.
- [20] R. Majumdar, A. Mathur, M. Pirron *et al.*, "Paracosm: A test framework for autonomous driving simulations," in *International Conference on Fundamental Approaches to Software Engineering*. Springer, Cham, 2021, pp. 172–195.
- [21] D. Chen, B. Zhou, V. Koltun *et al.*, "Learning by cheating," in *Conference on Robot Learning*. PMLR, 2020, pp. 66–75.
- [22] S. Rayadurgam and M. Heimdahl, "Generating mc/dc adequate test sequences through model checking," in *28th Annual NASA Goddard Software Engineering Workshop, 2003. Proceedings.*, 2003, pp. 91–96.
- [23] T. Dreossi, D. J. Fremont, S. Ghosh *et al.*, "Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems," in *International Conference on Computer Aided Verification*. Springer, 2019, pp. 432–442.
- [24] B. Schütt, T. Braun, S. Otten *et al.*, "Sceml: A graphical modeling framework for scenario-based testing of autonomous vehicles," in *23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 2020, pp. 114–120.

- [25] W. Ding, B. Chen, M. Xu *et al.*, “Learning to collide: An adaptive safety-critical scenarios generating method,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 2243–2250.
- [26] K. Dalbey, M. Eldred, G. Geraci *et al.*, “Dakota a multilevel parallel object-oriented framework for design optimization parameter estimation uncertainty quantification and sensitivity analysis: Version 6.14 theory manual.” Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2021.
- [27] D. Zhao, X. Huang, H. Peng *et al.*, “Accelerated evaluation of automated vehicles in car-following maneuvers,” *IEEE Transactions on Intelligent Transportation Systems*, 2017.
- [28] J. H. Halton, “On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals,” *Numerische Mathematik*, vol. 2, no. 1, pp. 84–90, 1960.
- [29] C. E. Tuncali, T. P. Pavlic, and G. Fainekos, “Utilizing s-talro as an automatic test generation framework for autonomous vehicles,” in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2016, pp. 1470–1475.
- [30] C. E. Tuncali, G. Fainekos, H. Ito *et al.*, “Simulation-based adversarial test generation for autonomous vehicles with machine learning components,” in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1555–1562.
- [31] A. Prakash, K. Chitta, and A. Geiger, “Multi-modal fusion transformer for end-to-end autonomous driving,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.
- [32] D. Chen, V. Koltun, and P. Krährenbühl, “Learning to drive from a world on rails,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 15 590–15 599.
- [33] I. Dejanović, R. Vadera, G. Milosavljević *et al.*, “Textx: a python tool for domain-specific languages implementation,” *Knowledge-Based Systems*, vol. 115, pp. 1–4, 2017.
- [34] O. Hummel and C. Atkinson, “The managed adapter pattern: Facilitating glue code generation for component reuse,” in *Formal Foundations of Reuse and Domain Engineering*, S. H. Edwards and G. Kulczycki, Eds. Springer Berlin Heidelberg, 2009, pp. 211–224.
- [35] O. CARLA, “Carla leaderboard challenge,” 2020. [Online]. Available: <https://leaderboard.carla.org/>
- [36] C. Hartsell, N. Mahadevan, S. Ramakrishna *et al.*, “Model-based design for cps with learning-enabled components,” in *Proceedings of the Workshop on Design Automation for CPS and IoT*, ser. DESTION ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–9. [Online]. Available: <https://doi.org/10.1145/3313151.3313166>