ICSE: G: Test Scenario Generation for Autonomous Driving Systems with Reinforcement Learning

Chengjie Lu
Simula Research Laboratory and University of Oslo
Oslo, Norway
chengjielu@simula.no

***Abstract*—We have seen rapid development of autonomous driving systems (ADSs) in recent years. These systems place high requirements on safety and reliability for their mass adoption, and ADS testing is one crucial approach to ensure their success. However, it is impossible to test all the scenarios due to the inherent complexity and uncertainty of ADSs and their driving tasks. Besides, the operating environment of ADSs is dynamic, continuously evolving, and full of uncertainties, which requires a testing approach adaptive to the environment. Reinforcement learning (RL) has shown great potential in various complex tasks requiring constant adaptation to dynamic environments. To this end, this paper presents *RLTester*, a novel ADS testing approach, that adopts reinforcement learning (RL) to learn critical environment configurations (i.e., test scenarios) of the operating environment of ADSs that could reveal their unsafe behaviors. To generate diverse and critical test scenarios, we defined 142 environment configuration actions and adopted the *Time-To-Collision* metric to construct the reward function. Our evaluation shows that *RLTester* discovered a total of 256 collisions, of which 192 are unique, and took an average of 11.59 seconds for each collision. Further, *RLTester* is effective in generating more diverse test scenarios compared to a state-of-the-art approach, *DeepCollision*. We also introduce an open-source driving scenario dataset, DeepScenario, which consists of over 30K driving scenarios.**

***Index Terms*—Autonomous Driving System Testing, Critical Scenario, Reinforcement Learning, Scenario Dataset**

I. Problem and Motivation

Problem. In recent years, we have seen rapid development of autonomous driving systems (ADSs), which are cyber-physical systems capable of sensing the environment and making decisions autonomously [3]. However, due to the complexity of ADSs themselves and the complexity of their operating environments, the number of possible scenarios for testing ADSs is infinite. Further, their operating environments are dynamic, continuously evolving, and full of various uncertainties, (e.g., unexpected pedestrian behaviors) and ADSs must operate safely in such operating environments. Therefore, it's important to test ADSs to ensure their dependability under various driving/test scenarios [4]. Thus, it's important to have a test approach that can adapt to the changes in the operating environment and generate critical environment configurations, under which an autonomous vehicle must be tested in terms of its ability to operate safely.

Online testing is an important ADS testing method aiming at identifying system-level failures by generating critical test scenarios, in which an ADS is embedded in an operating environment (which is often a simulated virtual environment) and tested when it interacts with the environment. Several *online* testing approaches [5], [6], [7], [8] have been proposed by applying various techniques such as search-based testing (SBT) and reinforcement learning (RL). For example, SBT has been widely used to generate test scenarios by formulating a test generation problem as an optimization problem, which can be addressed with meta-heuristic optimization algorithms (e.g., genetic algorithms [9]). SBT approaches [10], [11], [12] have shown promising performance in identifying system-level failures, however, it remains challenging to generate critical scenarios cost-effectively due to the huge computational and time overhead caused by evaluating test scenarios. Additionally, existing SBT approaches show limited effectiveness when testing a long-term decision-making task under a dynamic and continuously changing environment [13].

Motivation. Recently, RL has demonstrated great potential in various challenging problems requiring adaption to a continuously changing environment [13]. Several RL-based approaches [7], [8], [14] have been proposed and have shown promising results for testing ADSs. However, covering as many test scenarios as possible is challenging, with one of the reasons being insufficient coverage of the configurable environment parameters. For example, Chen et al. [8] targeted lane-change scenarios and controlled three types of adversarial vehicles with only longitudinal movements. *DeepCollision* [7] aims at four driving tasks and adopts **52** environment configuration actions. To generate critical scenarios, *DeepCollision* employs *collision probability* to design the reward function.

This paper presents *RLTester*, an RL-based *online* testing approach, which extended *DeepCollision* in terms of the ability to cover more *diverse* and *critical* test scenarios. The main contributions are: 1) We expanded the number of environment configurable parameters and defined **142** actions for configuring the environment; 2) We proposed a novel reward function design based on a commonly used safety indicator (i.e., *Time-To-Collision* [15]). The results show that *RLTester* outperformed *DeepCollision* in terms of generating more diverse and critical scenarios.

This SRC Grand Final submission summarizes some key findings of the paper [1], which has been submitted to ACM Transactions on Software Engineering and Methodology (TOSEM) and is currently undergoing major revision. The dataset, i.e., DeepScenario [2], has been accepted and published in the 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR).

II. Background and Related Work

A. Reinforcement Learning

Reinforcement learning is about agents learning optimal behavioral policies to achieve goals through iterative interactions with unknown environments [13]. Specifically, at each learning step, an RL agent observes the environment state and decides an action to take based on its current behavioral policy, which is the mapping between states to actions. After taking the action, the performance of the agent is evaluated with a reward, based on which the behavioral policy will be updated. The goal is to maximize the cumulative reward of a long-term decision-making process. **Deep Q-Learning (DQN)** [16] is a classical RL algorithm that has demonstrated good performance in solving complicated problems. The behavioral policy in DQN is constructed as a deep neural network (DNN, also known as Q-Network) that takes states as inputs and selects optimal actions based on the network's predictions. The application of DNN has made DQN successful in autonomous driving [17].

B. Online ADS Testing

Online testing approaches have been proposed to test ADSs in a simulated/physical operating environment. Various techniques have been applied in these approaches, including SBT and RL. SBT typically focuses on the test scenario generation with the guidance of optimization objectives such as violating safety requirements [11], minimizing time to collision [10], and maximizing the speed at collision [5]. For example, Abdessalem et al. [5] proposed NSGAII-DT, which considers two objectives (i.e, speed at the time of collision, and distance to obstacles) to generate critical scenarios for vision-based control systems by combining NSGA-II with decision tree classification models. SBT approaches have shown promising performance in identifying system-level failures, however, these approaches remain challenging and show limited effectiveness when testing a long-term decision-making task under a dynamic environment [13]. RL-based techniques identify critical scenarios by adaptively exploring the vast scenario space. DeepCollision [7] is an RL-based approach that generates safety-critical scenarios by dynamically configuring an AV's operating environment. By combining RL and multi-objective search, Haq et al. [18] proposed a multi-objective RL approach, MORLOT, for testing AV. MORLOT uses RL to adaptively generate critical scenarios that can cause requirement violations and adopts multi-objective search to cover as many requirements as possible. However, covering as many test scenarios as possible is challenging, with one of the reasons being insufficient coverage of the configurable environment parameters. Therefore, in this paper, we propose an RL-based *online* ADS testing approach, which aims to cover more *diverse* and *critical* test scenarios.

III. RLTester Methodology

RLTester generates critical test scenarios through automatically learning and dynamically configuring the operating environment of an autonomous vehicle under test (i.e., AVUT) with RL. As illustrated in Fig. 1, *RLTester* consists of three main

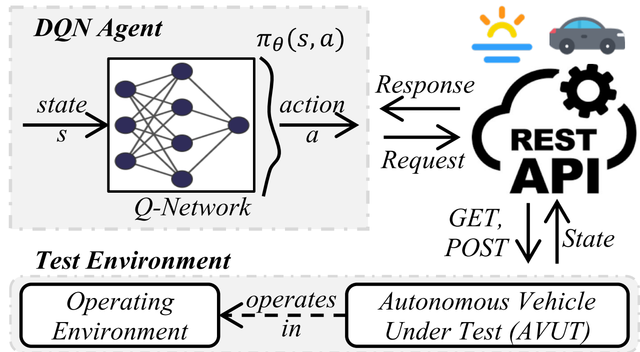

Fig. 1. Overview of *RLTester*

components: *Test Environment* where an *AVUT* operates in its *Operating Environment*; a list of *REST APIs* for configuring the environment and obtaining its states; and *DQN Agent* that generates actions to configure *Operating Environment*. *DQN Agent* first observes a *state* s and decides an *action* a based on s. Such an action a is implemented as an *REST API*, which introduces new configurations to the environment through an *HTTP Request*. After a's execution, both *AVUT* and its *Operating Environment* enter a new state s', which is returned to *DQN Agent* via an *HTTP Response*. The agent's performance on taking action a is evaluated with a *reward* r. Below is a detailed description of each component.

A. Test Environment

RLTester generates critical test scenarios in a simulated *Test Environment*. To build it, *RLTester* employs SVL Simulator 2021.1 [19] to simulate the *AVUT* and its *Operating Environment* and deploys the ADS Apollo 5.0 [20] on the *AVUT* to enable driving. Specifically, we selected the San Francisco HD map based on the South of Market (SoMa) district in San Francisco. As for the *AVUT*, we employed Lincoln2017MKZ, a four-door Sedan commonly applied for autonomous driving research [21], [22].

B. Environment Configuration REST API

When testing ADS in a simulated environment, we assume that the more environmental parameters to be manipulated imply more diverse scenarios that can be generated. Hence, we expanded *DeepCollision* and obtained three types of parameters, for *Static Objects*, *Dynamic Objects*, and *Weather&Time*. For example, we add an additional parameter (i.e., *color*) for *NPC Vehicle* since evidence has shown the influence of obstacle color on the decision-making of the *AVUT* [23]. Invocations of these parameters have been implemented as **142** *REST APIs* so that we can configure the *Operating Environment* and obtain the status of the *Test Environment* through HTTP requests and responses.

C. DQN Agent

To learn environment configurations with RL, *RLTester* adopts the following *state encoding*, *action space*, and *reward function* definitions such that DQN can be applied. Regarding *state encoding*, we adopt multi-modal sensor fusion [24] as the encoding strategy, which encodes a state using camera images, Lidar bird's eye view representation, and *AVUT*'s state measurements (i.e., speed). After a state is observed, it is fed into the *Q-Network* for feature extraction, based on the results, the *DQN Agent* will decide an action to configure the environment. We designed the *Q-Network* as a multi-modal sensor fusion network considering its promising performance in various autonomous driving perception tasks [25]. Specifically, the *Q-Network* consists of an image-processing CNN module, a lidar-processing CNN module, and a fully connected *AVUT* state measurement-processing module. The action space is composed of environment configuration REST APIs, and after the invocation of action, we will get a set of outputs, based on which we define the reward function for *RLTester*. Specifically, we adopt *Time-To-Collision (TTC)* as the output, which is a commonly used safety indicator for measuring the severity of traffic conflicts [15]. The smaller a *TTC* value, the higher the severity of the traffic conflict. Recall that *RLTester* aims to generate critical scenarios that can lead to more safety violations. Thus, we define a reward function $\mathrm{R}_{\mathrm{TTC}}$, which encourages the *DQN Agent* to take an environment configuration action that can minimize *TTC* so that the traffic conflicts can be maximized.

IV. Evaluation

To study the effectiveness of *RLTester* in terms of generating critical scenarios, we first compared *RLTester* with two baselines adopted in *DeepCollision*, i.e., Random Strategy (*RS*), which randomly selects actions to configure the environment, and Greedy Strategy (*GS*), which greedily selects an action that can achieve the best performance in terms of *TTC*. We then compared *RLTester* with *DeepCollision* in terms of covering more configurable environment parameters and generating more diverse and critical scenarios. The experiments were executed on four roads (R1...R4) selected based on different road structures and characteristics, and all experiments were repeated 20 times.

Comparison with *RS* and *GS* regarding the number of discovered collisions ($\#\mathcal{C}$) and the time cost for discovering a collision ($\mathcal{T}_{\mathcal{C}}$). The results show that, for $\#\mathcal{C}$, *RLTester* discovered a total of 256 collisions, which outperformed *RS* (i.e., 41) and *GS* (i.e., 73). As for $\mathcal{T}_{\mathcal{C}}$, the results show that *RLTester* took an average of 11.59 seconds to discover a collision, outperforming *RS* (16.86 seconds) and *GS* (19.05 seconds). In addition, we also performed statistical tests to compare the results of 20 executions. Specifically, we used the Mann and Whitney test for assessing statistical significance and calculated Vargha and Delaney metric $\hat{A}_{12}$ for the effect size. The statistical results show that *RLTester* significantly outperformed *RS* and *GS* in terms of $\#\mathcal{C}$ and $\mathcal{T}_{\mathcal{C}}$, indicating that *RLTester* is effective in generating more critical test scenarios within less time.

Comparison with *DeepCollision* with two *diversity* metrics: $DIV_{API} = \#U_{API}/T_{API}, DIV_{SC} = \#U_{SC}/T_{SC}$, where, $\#U_{API}$ and $\#U_{SC}$ are the number of *unique* API calls and *unique* test

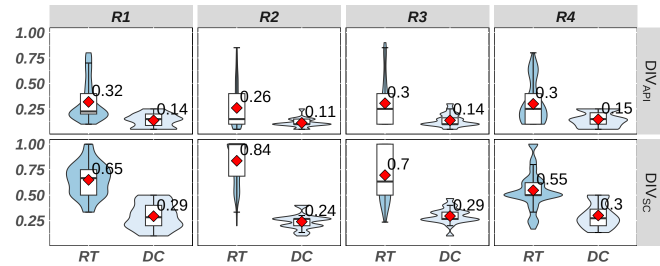

Fig. 2. Descriptive statistics of Diversity Metrics (DIV_{API}: *API Diversity*, DIV_{SC}: *Scenario Diversity*; *RT*: *RLTester*, *DC*: *DeepCollision*)

scenarios; T_{API} and T_{SC} are the total number of API calls and test scenarios. As shown in Fig. 2, *RLTester* outperformed *DeepCollision* regarding DIV_{API} and DIV_{SC} for all the roads. In terms of $\#\mathcal{C}$ and $\mathcal{T}_{\mathcal{C}}$, *RLTester* outperformed *DeepCollision* as it discovered 192 *unique* collisions with an average $\mathcal{T}_{\mathcal{C}}$ of 12 seconds. Instead, *DeepCollision* discovered a total of 288 collisions (with only 40 are *unique*) taking an average $\mathcal{T}_{\mathcal{C}}$ of 18.44 seconds. The differences with *DeepCollision* are all statistically significant, indicating that *RLTester* effectively covers more diverse environment parameters and generates more diverse test scenarios. After replaying the scenarios, we found that *DeepCollision*'s lower effectiveness in *diversity* is due to calling the same few APIs repeatedly. For example, most collisions occur when pedestrians cross the road.

V. Scenario Dataset

We created an open-source driving scenario dataset, DeepScenario [2], by collecting the test scenarios generated by *RLTester*. DeepScenario contains more than *30K* executable driving scenarios that can be utilized for various autonomous driving research, such as system-level ADS testing, test selection and optimization, and regression tests. We proposed a parameter-based scenario definition language. In addition, we developed a scenario toolset to facilitate the collection, replay, and usage of scenarios. We discuss the scenario definition, toolset, and dataset usage as follows:

A. Scenario Definition

According to Ulbrich et al. [26], a driving scenario S describes the temporal development between several scenes. A scene describes a snapshot of the environment. In DeepScenario, a driving scenario S is defined as a tuple with several scenes: $S = < scene_1, scene_2, ..., scene_n, T >$, where T is the time that S spans and n denotes the number of scenes in S. We further define a scene sc as a 3-tuple: sc = <*ego story*, *obstacle story*, *environment state*>, where 1) *ego story* denotes the operations and kinetic parameters of the *AVUT*; 2) *obstacle story* denotes the operations and kinetic parameters of the static and dynamic obstacles such as NPC vehicles and pedestrians; 3) *environment state* includes weather conditions, time of day, and the driving tasks. Additionally, we developed a Domain Specific Language (DSL) for scenario representation based on the scenario definition. The scenario file extension is $.deepscenario$, an XML-based file extension.

B. Scenario Toolset

We developed *ScenarioCollector* to automatically collect driving scenarios. *ScenarioCollector* has already been integrated into the environment configuration framework and can collect and store the driving scenarios in scenario file format. *ScenarioRunner* was developed to support replaying driving scenarios by taking scenario files as inputs. Specifically, it has two ways of replaying a driving scenario. First, it can exactly replay the behaviors of the ego vehicle and obstacles by reading their kinetic parameters. By selecting and replaying driving scenarios in this way, *ScenarioRunner* can facilitate further analysis and diagnoses. Furthermore, *ScenarioRunner* can integrate different *ADSs*. In this way, the behaviors of the ego vehicle are not replayed by *ScenarioRunner* but controlled by the *ADS*, and the behaviors of obstacles can still be accurately replayed. This way enables testing various *ADSs* by integrating ADSs in the replayed driving conditions.

C. Driving Scenario Attributes

To characterize driving scenarios in the dataset, we associate each scenario with six attributes, which are calculated based on the test results of driving scenarios. Specifically, for a scenario $\mathcal{S}$, its attributes can be classified into two types defined as follows: ***Performance Attributes*** are related to the safety/comfort measures, and Time-To-Collision (*TTC*), Distance-to-Obstacles (*DTO*), and *Jerk* are the three reward attributes, which measure the extent of safety/comfort when driving in $\mathcal{S}$. Specifically, smaller *TTC*/*DTO* values indicate higher safety risk, while larger *Jerk* values indicate less comfort. ***Collision Attributes*** are collision-related attributes. We have defined three collision attributes: *Collision (COL)* is a Boolean attribute indicating if the ego vehicle collided with obstacles in $\mathcal{S}$. *Collision-Type (COLT)* is an enumerated attribute that shows the type of obstacle the ego vehicle collided with. Concretely, *COLT* has three possible values, which are *Non-player character (NPC) Vehicle*, *Pedestrian*, and *Static Obstacle*. *Speed-At-Collision (SAC)* is an attribute that records the speed at which the ego vehicle in $\mathcal{S}$ collided (if it happened) with the obstacle.

D. Dataset Usage

Testing ADSs to detect system-level failures. By running scenarios with *ScenarioRunner*, we can deploy an ADS in the environment and perform testing to identify system-level failures of the ADS. Moreover, various ADSs or various ADS versions can be integrated with *ScenarioRunner*, which can be tested by executing scenarios.

Analyzing scenarios for further diagnoses. With the driving scenarios attributes, we can easily select critical scenarios from the dataset that caused higher safety/comfort risk or collisions of the ego vehicle. The selected scenarios can be replayed with *ScenarioRunner* to facilitate further analysis and diagnoses of ADSs. For example, we can analyze collision scenarios concerning collision type and identify key environmental factors for different types of collisions.

Selecting and prioritizing scenarios for regression testing. In practice, generating critical testing scenarios is very expensive in terms of time costs and computational resources, and as ADS evolves, regression testing for multiple versions will become even more expensive [27]. By using search-based selection techniques, we can further select critical scenarios from DeepScenario and prioritize them to support regression testing of ADSs.

The dataset is available on our GitHub online repository[1] and Zenodo [28].

VI. Discussion and Lesson Learned

A. Incorporating Multi-objective RL

Currently, *RLTester* uses one objective to design the reward function. In practice, many requirements, such as safety and functional requirements, need to be tested simultaneously. This motivates us to explore different designs of *RLTester* to support multi-objective driving scenario generation. One solution is to merge multiple objectives into a single objective with a weighted sum and use it to design a reward function. Another solution is to employ multi-objective reinforcement learning (MORL) [29], which aims to find a policy that can optimize multiple objectives simultaneously; thereby doesn't require merging multiple objectives into a single objective. We will investigate the multi-objective *RLTester* in the future.

B. Handling Infinite Scenario Space

We designed 142 REST APIs by considering various environmental parameters and their possible values. However, a test scenario is characterized by many parameters, and the possible combinations of these parameters are theoretically infinite. Therefore, it's important to cost-effectively explore the huge scenario space and pay more attention to those environmental parameters that have more importance in the critical scenario generation. To this end, in our recent work, we proposed a novel testing approach, EpiTESTER [30], focusing on finding critical scenarios efficiently. In particular, EpiTESTER uses an epigenetic algorithm [31] that can selectively express or silence certain environmental parameters, thereby enabling the search to focus on the most important parameters and reduce the search space. To decide the expression of parameters, we employed the attention mechanism [32] that can adaptively determine the parameter's contribution to critical scenarios. In the future, we will investigate integrating EpiTESTER with *RLTester*, such as designing the *Q-Network* with the attention mechanism.

VII. Concluding Remarks

We present *RLTester*, an RL-based ADS testing approach that generates more unique and critical scenarios in less time. Moreover, we present an open-source driving scenario dataset, DeepScenario. Future works include a systematic definition of environmental parameter coverage and scenario coverage criteria, using multi-objective reinforcement learning, and integrating EpiTESTER into the overall approach.

[1]https://github.com/Simula-COMPLEX/DeepScenario

REFERENCES

[1] C. Lu, T. Yue, M. Zhang, and S. Ali, "Deepqtest: Testing autonomous driving systems with reinforcement learning and real-world weather data," *arXiv preprint arXiv:2310.05170*, 2023.

[2] C. Lu, T. Yue, and S. Ali, "Deepscenario: An open driving scenario dataset for autonomous driving system testing," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, pp. 52–56, 2023.

[3] A. Stocco, B. Pulfer, and P. Tonella, "Mind the gap! a study on the transferability of virtual vs physical-world testing of autonomous driving systems," *IEEE Transactions on Software Engineering*, 2022.

[4] 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.

[5] R. B. Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing vision-based control systems using learnable evolutionary algorithms," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pp. 1016–1026, IEEE, 2018.

[6] A. Calò, P. Arcaini, S. Ali, F. Hauer, and F. Ishikawa, "Generating avoidable collision scenarios for testing autonomous driving systems," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pp. 375–386, IEEE, 2020.

[7] C. Lu, Y. Shi, H. Zhang, M. Zhang, T. Wang, T. Yue, and S. Ali, "Learning configurations of operating environment of autonomous vehicles to maximize their collisions," *IEEE Transactions on Software Engineering*, 2022.

[8] B. Chen, X. Chen, Q. Wu, and L. Li, "Adversarial evaluation of autonomous vehicles in lane-change scenarios," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[9] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.

[10] R. Ben Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing advanced driver assistance systems using multi-objective search and neural networks," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pp. 63–74, 2016.

[11] R. B. Abdessalem, A. Panichella, S. Nejati, L. C. Briand, and T. Stifter, "Testing autonomous cars for feature interaction failures using many-objective search," in *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 143–154, IEEE, 2018.

[12] C. Gladisch, T. Heinz, C. Heinzemann, J. Oehlerking, A. von Vietinghoff, and T. Pfitzer, "Experience paper: Search-based testing in automated driving control applications," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 26–37, IEEE, 2019.

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[14] F. U. Haq, D. Shin, and L. Briand, "Many-objective reinforcement learning for online testing of dnn-enabled systems," *arXiv preprint arXiv:2210.15432*, 2022.

[15] R. Van Der Horst and J. Hogema, "Time-to-collision and collision avoidance systems," 1993.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[17] Z. Ruiming, L. Chengju, and C. Qijun, "End-to-end control of kart agent with deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 1688–1693, IEEE, 2018.

[18] F. U. Haq, D. Shin, and L. C. Briand, "Many-objective reinforcement learning for online testing of dnn-enabled systems," in *Proceedings of the 45th International Conference on Software Engineering*, ICSE '23, p. 1814–1826, IEEE Press, 2023.

[19] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta, *et al.*, "Lgsvl simulator: A high fidelity simulator for autonomous driving," in *2020 IEEE 23rd International conference on intelligent transportation systems (ITSC)*, pp. 1–6, IEEE, 2020.

[20] Baidu, "Apollo: An open autonomous driving platform." https://github.com/ApolloAuto/apollo.

[21] S. Xu, H. Peng, Z. Song, K. Chen, and Y. Tang, "Design and test of speed tracking control for the self-driving lincoln mkz platform," *IEEE Transactions on Intelligent Vehicles*, vol. 5, no. 2, pp. 324–334, 2019.

[22] V. Vegamoor, S. Rathinam, and S. Darbha, "String stability of connected vehicle platoons under lossy v2v communication," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 8834–8845, 2021.

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

[24] Z. Huang, C. Lv, Y. Xing, and J. Wu, "Multi-modal sensor fusion-based deep neural network for end-to-end autonomous driving with scene understanding," *IEEE Sensors Journal*, vol. 21, no. 10, pp. 11781–11790, 2020.

[25] D. Feng, C. Haase-Schütz, L. Rosenbaum, H. Hertlein, C. Glaeser, F. Timm, W. Wiesbeck, and K. Dietmayer, "Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1341–1360, 2020.

[26] S. Ulbrich, T. Menzel, A. Reschka, F. Schuldt, and M. Maurer, "Defining and substantiating the terms scene, situation, and scenario for automated driving," in *2015 IEEE 18th international conference on intelligent transportation systems*, pp. 982–988, IEEE, 2015.

[27] C. Lu, H. Zhang, T. Yue, and S. Ali, "Search-based selection and prioritization of test scenarios for autonomous driving systems," in *International Symposium on Search Based Software Engineering*, pp. 41–55, Springer, 2021.

[28] C. Lu, T. Yue, and S. Ali, "DeepScenario: An Open Driving Scenario Dataset for Autonomous Driving System Testing," Mar. 2023.

[29] C. Liu, X. Xu, and D. Hu, "Multiobjective reinforcement learning: A comprehensive overview," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 3, pp. 385–398, 2014.

[30] C. Lu, S. Ali, and T. Yue, "Epitester: Testing autonomous vehicles with epigenetic algorithm and attention mechanism," *arXiv preprint arXiv:2312.00207*, 2023.

[31] D. H. Stolfi and E. Alba, "Epigenetic algorithms: A new way of building gas based on epigenetics," *Information Sciences*, vol. 424, pp. 250–272, 2018.

[32] 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.