Test Scenario Generation for Autonomous Driving Systems with Reinforcement Learning

Chengjie Lu
Department of Engineering Complex Software Systems
Simula Research Laboratory
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 of the crucial approaches to ensure the success of ADSs. To this end, this paper presents *RLTester*, a novel ADS testing approach, which 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 collisions, and took on average 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*.**

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

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 [1]. 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, 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 [2].

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 and tested when it interacts with the environment. Several *online* testing approaches [3], [4], [5], [6] have been proposed by applying various techniques such as search-based testing (SBT) and reinforcement learning (RL). SBT approaches [7], [8], [9] have shown promising performance in identifying system-level failures, however, existing SBT approaches show limited effectiveness when testing a long-term decision-making task under a dynamic environment [10].
Motivation. Recently, RL has demonstrated great potential in various challenging problems requiring adaption to a continuously-changing environment [10]. Several RL-based approaches [5], [6], [11] 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. [6] targeted lane-change scenarios and controlled three types of adversarial vehicles with only longitudinal movements. *DeepCollision* [5] 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* [12]). The results show that *RLTester* outperformed *DeepCollision* in terms of generating more diverse and critical scenarios.

II. Background

Reinforcement learning (RL) is about agents learning optimal behavioral policies to achieve goals through iterative interactions with unknown environments [10]. 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)** [13] 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 (Q-Network) that takes states as inputs and selects optimal actions based on the network's predictions.

III. RLTester Methodology

As illustrated in Fig. 1, *RLTester* consists of three main components: *Test Environment* where an *AVUT* operates in its

This work is supported by the Co-evolver project (No. 286898/F20) funded by the Research Council of Norway.

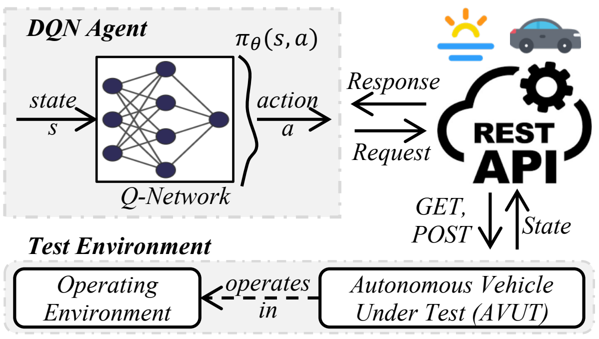

Fig. 1. Overview of *RLTester*

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.

Test Environment. *RLTester* generates critical test scenarios in a simulated *Test Environment*. To build it, *RLTester* employs SVL Simulator 2021.1 [14] to simulate the *AVUT* and its *Operating Environment* and deploys the ADS Apollo 5.0 [15] on the *AVUT* to enable driving.

Environment Configuration REST API. When testing ADS in a simulated environment, the more environmental parameters to be manipulated implies more diverse scenarios that can be generated. Hence, we expand *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*. Invocations of these parameters have been implemented as **142** *REST APIs*.

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 [16] as the encoding strategy, which encodes a state using camera images, Lidar point clouds, and *AVUT*'s speeds. After a state is observed, it is fed into the *Q-Network* for feature extraction, based on the results of which the *DQN Agent* will decide an action to configure the environment. The action space is composed of environment configuration REST APIs, and after the invocation of an 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 [12]. The smaller a *TTC* value, the higher the severity of the traffic conflict. Then we define a reward function $\mathrm{R_{TTC}}$, which encourages the *DQN Agent* to take an environment configuration action that can minimize *TTC*.

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*) and Greedy Strategy (*GS*). 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), 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 $\mathcal{T}_\mathcal{C}$.

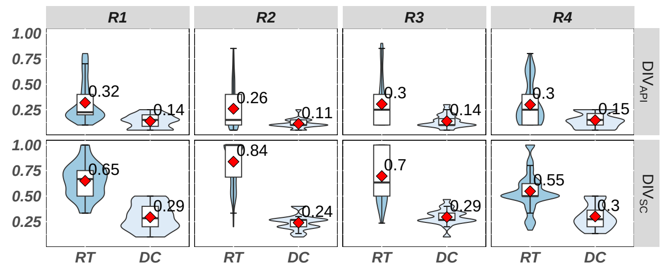

Fig. 2. Descriptive statistics of Diversity Metrics (*DIV_{API}*: *API Diversity*, *DIV_{SC}*: *Scenario Diversity*; *RT*: *RLTester*, *DC*: *DeepCollision*)

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 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 statistical significant. 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 occurred when pedestrians crossed the road.

V. Concluding Remarks

We present *RLTester*, an RL-based ADS testing approach that generates more unique and critical scenarios with less time. Future works include a systematic definition of environmental parameter coverage and scenario coverage criteria, and analyses of scenarios from various aspects, e.g., safety and comfort.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

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

[13] 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.

[14] 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.

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

[16] 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.